TAMPEREEN TEKNILLINEN YLIOPISTO

MIIKKA RAIVIO

ALUSTARIIPPUMATTOMAN MOBIILISOVELLUSKEHITYKSEN TEKNIIKAT

Diplomityö

Tarkastaja: professori Tommi Mikkonen
Tarkastaja ja aihe hyväksytty
Tieto- ja sähkötekniikan tiedekuntaneuvoston kokouksessa 6. helmikuuta 2013

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
RAIVIO, MIIKKA: Alustariippumattoman mobiilisovelluskehityksen tekniikat
Diplomityö, 62 sivua, 2 liitesivua
Joulukuu 2013
Pääaine: Ohjelmistotuotanto
Tarkastaja: professori Tommi Mikkonen
Avainsanat: alustariippumaton, mobiilisovellus, sovelluskehitys, natiivisovellukset, tulkattavat sovellukset, hybridisovellukset, Appcelerator Titanium, PhoneGap, Sencha Touch

Mobiilisovellusten kehitys mobiilikäyttöjärjestelmien virallisten kehitystyökalujen avulla muuttuu haastavaksi, kun sama sovellus on toteutettava useammalle kuin yhdelle kohdealustalle. Tällöin ohjelmistokehittäjän on useimmiten hallittava kullekin mobiilikäyttöjärjestelmälle ominaiset sovelluskehitys- ja suunnitteluperiaatteet, mikä tarkoittaa tuettavien alustojen määrään nähden yhtä usean sovelluskehitysprojektin läpikäymistä.

Tässä diplomityössä tutustuttiin alustariippumattoman mobiilisovelluskehityksen tekniikoihin ja työkaluihin, jotka väitetysti nopeuttavat usealle alustalle kohdistuvaa sovelluskehitystä hyödyntämällä mahdollisimman paljon samaa lähdekoodia tuettavien kohdealustojen välillä. Tämän diplomityön tavoitteena oli lisäksi löytää Metson Mining and Construction -segmentin alaisuudessa toimivan mobiilisovelluskehitystiimin käyttötarkoituksiin parhaiten soveltuva alustariippumaton kehitystyökalu. Tätä varten valittiin kolme lupaavinta kehitystyökalua, Appcelerator Titanium, PhoneGap ja Sencha Touch, joiden avulla rakennettiin esimerkkisovellus Android- ja iOS-kohdealustoille. Eri työkaluilla rakennettujen esimerkkisovellusten toteutusprosesseja ja lopputuloksia vertailtiin yhdeksän ennalta määritetyn arviointikriteerin avulla.

Tutkimuksessa saatujen tulosten perusteella voitiin todeta, että alustariippumattomat mobiilisovelluskehitystyökalut ovat varteenotettava vaihtoehto virallisille alustakohtaisille kehitystyökaluille. Niiden oppimiskynnystä pidettiin alustojen virallisia kehitystyökaluja matalampana ja kehitysvauhtia nopeampana usealle alustalle kohdistuvassa sovelluskehityksessä. Alustariippumattomien kehitystyökalujen suurimpina heikkouksina alustojen virallisiin kehitystyökaluihin nähden pidettiin niitä rajatumpia laiteominaisuuksille tarjottuja ohjelmointirajapintoja sekä myöhäisempää pääsyä alustojen uusiin toiminnallisuuksiin. Tämän tutkimuksen johtopäätöksissä esitettiin suuntaviivat oikean mobiilisovelluskehitystyökalun valintaan erityyppisissä mobiilisovelluskehitysprojekteissa.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
RAIVIO, MIIKKA: Technologies of mobile cross-platform application development
Master of Science Thesis, 62 pages, 2 Appendix pages
December 2013
Major: Software Engineering
Examiner: Professor Tommi Mikkonen
Keywords: cross-platform, mobile application, application development, native applications, interpreted applications, hybrid applications, Appcelerator Titanium, PhoneGap, Sencha Touch

Developing mobile applications with the official platform specific software developments kits (SDKs) becomes challenging when the same application has to be created for multiple mobile operating systems. The application developer has to master application development and design practices specific to each mobile operating system, which often results to separate development projects for each supported platform.

This thesis studied the technologies and tools for mobile cross-platform application development. Cross-platform tools (CPTs) are advertised to speed up the application development targeting multiple mobile platforms by sharing the code base between the platforms. In addition, this thesis aimed to find the most suitable CPT for a mobile development team working for Mining and Construction segment of Metso. Three most promising tools, Appcelerator Titanium, PhoneGap and Sencha Touch, were selected and a test application was development with them for Android and iOS platforms. The development process and the outcome of the test application developed with the selected CPTs were compared with the help of nine predetermined evaluation criteria.

According to the results of this research, the mobile CPTs proved to be a viable alternative for the official platform specific SDKs. Their learning curve was considered steeper and application development speed higher when targeting multiple platforms. The most notable drawbacks of using CPTs were more limited application programming interfaces (APIs) to device features and delayed access to the latest functionality of the supported mobile operating systems. In the conclusions of this research, guidelines for selecting the right mobile application development tool for different types of mobile application development projects were proposed.

ALKUSANAT

Tämä diplomityö tehtiin Metso Minerals Oy:lle itsenäisenä projektina muiden työtehtävien ohella. Työn ohjaajana toimi Metson puolelta Esko Harjama, jota haluan kiittää erittäin mielenkiintoisesta diplomityöaiheesta, työn ohjaamisesta ja kommentoinnista sekä työn tekemiseen järjestetystä ajasta. TTY:n puolelta työn tarkastajana toimi professori Tommi Mikkonen, jota haluan kiittää työn ohjauksesta, arvokkaista kommenteista ja erityisesti kirjoitusmotivaationi ylläpidosta. Esitän kiitokset myös perheelleni ja ystävilleni, jotka olivat tukenani sekä opintojen että diplomityön kirjoittamisen aikana.

Tampereella 12.11.2013

_____

Miikka Raivio

SISÄLLYS

Tiivistelmä II
Abstract III
Alkusanat IV
1. Johdanto 1
2. Sovelluskehitys mobiililaitteille 3
2.1. Mobiililaitteet ja -sovellukset 3
2.2. Mobiilikäyttöjärjestelmät 3
2.2.1. Markkinatilanne 3
2.2.2. Android 4
2.2.3. iOS 6
2.2.4. Windows Phone 8
2.2.5. Muut kilpailijat 9
2.3. Mobiililaitteet Metsolla 9
3. Alustariippumaton mobiilisovelluskehitys 13
3.1. Tarve alustariippumattomalle mobiilisovelluskehitykselle 13
3.2. Web-sovellukset 15
3.3. Alustariippumattomat kehitystyökalut 18
4. Alustariippumattomien kehitystyökalujen valinta- ja arviointikriteerit 23
4.1. Minimivaatimukset alustariippumattomalle kehitystyökalulle 23
4.1.1. Kohdealustat 23
4.1.2. Laiteominaisuudet ja muut toiminnallisuudet 24
4.2. Kehitystyökalujen vertailu 26
4.3. Kehitystyökalujen valinta 29
4.4. Kehitystyökalujen arviointikriteerit 30
5. Alustariippumattomien kehitystyökalujen testaus 32
5.1. Testialusta, testilaitteet ja web-palvelu 32
5.2. Sencha Touch 33
5.2.1. Kuvaus 33
5.2.2. Asennus ja käyttöönotto 34
5.2.3. Sovelluksen toteutus 34
5.2.4. Sovelluksen jakelu eri mobiilialustoille 36
5.3. PhoneGap 37
5.3.1. Kuvaus 37
5.3.2. Asennus ja käyttöönotto 37
5.3.3. Sovelluksen toteutus 38
5.3.4. Sovelluksen jakelu eri mobiilialustoille 38
5.4. Appcelerator Titanium 39
5.4.1. Kuvaus 39
5.4.2. Asennus ja käyttöönotto 39
5.4.3. Sovelluksen toteutus 39

5.4.4. Sovelluksen jakelu eri mobiilialustoille 42
6. Alustariippumattomien kehitystyökalujen arviointi 43
6.1. Määrälliset arviointikriteerit 43
6.1.1. Sovelluksen asennuspaketin koko 43
6.1.2. Sovelluksen keskusmuistin käyttö 44
6.1.3. Koodirivien lukumäärä 45
6.1.4. Koodin uudelleenkäytettävyys 46
6.1.5. Lisenssi 47
6.2. Laadulliset arviointikriteerit 48
6.2.1. Työkalun opittavuus 48
6.2.2. Työkalun dokumentaation kattavuus 49
6.2.3. Esimerkkisovelluksen onnistuminen 50
6.2.4. Esimerkkisovelluksen käyttöliittymän sulavuus 51
6.3. Yhteenveto 52
6.4. Jatkotutkimuksen kohteet 53
7. Johtopäätökset 55
Lähteet 58
Liite A: Kuvakaappaukset 63

1. JOHDANTO

Mobiililaitteiden nopea kehitys viimeisen kymmenen vuoden aikana on johtanut siihen, että yksinkertaisista taskuun mahtuvista kommunikointivälineistä on tullut täysiverisiä tietokoneita miljoonine ladattavine sovelluksineen. Viime vuosien aikana kuluttajamarkkinoilla valtaisan suosion saavuttaneiden kosketusnäytöllisten älypuhelinten ja tablettien potentiaali on huomioitu myös yritysmaailmassa.

Kansainvälisen prosessiteollisuuden teknologioita ja palveluita toimittavan Metson IT-strategiana on muuttua ajasta, paikasta ja laitteista riippumattomaksi yritykseksi edistämällä mobiiliteknologioiden ja -sovellusten käyttöönottoa. Metson Mining and Construction -segmentin (*lyh. MAC*) alaisuudessa toimivan Global Application Development -tiimin (*lyh. GAD*) tehtävänä on kehittää mobiilisovelluksia pääasiassa oman segmenttinsä työntekijöiden ja muiden sidosryhmien käyttöön. Pääpainona ovat olleet tähän asti iOS-mobiilikäyttöjärjestelmälle suunnatut sovellukset, mutta tuki muille alustoille alkaa olla ajankohtaista erityisesti Android-laitteiden yleistyessä.

Haasteena mobiilisovelluskehityksessä ovat mobiilikäyttöjärjestelmien toisistaan merkittävästi poikkeavat sovelluskehitys- ja suunnitteluperiaatteet. Jos sovelluskehittäjä haluaa rakentaa saman mobiilisovelluksen useammalle mobiilikäyttöjärjestelmälle, täytyy hänen useimmiten hallita kunkin mobiilikäyttöjärjestelmän omien kehitystyökalujen, ohjelmointikielen ja -rajapintojen lisäksi käyttöjärjestelmille ominaiset suunnittelu- ja tyyliperiaatteet. Saman sovelluksen toteuttaminen usealle mobiilikäyttöjärjestelmälle vaatii siten useimmiten yhtä usean sovelluskehitysprojektin läpikäymistä.

Mahdollinen ratkaisu tähän ongelmaan ovat alustariippumattomat kehitystyökalut, joiden avulla mobiilisovelluksia voidaan rakentaa ja jakaa usealle kohdealustalle hyödyntäen mahdollisimman paljon yhteistä sovelluskoodia alustojen välillä. Niiden tarkoituksena on vähentää kehitykseen kuluvaa aikaa ja kustannuksia usealle alustalle kohdistetussa sovelluskehityksessä verrattuna mobiilikäyttöjärjestelmien omien kehitystyökalujen avulla tehtävään sovelluskehitykseen.

Tämän diplomityön tarkoituksena on tutustua alustariippumattoman mobiilisovelluskehityksen tekniikoihin sekä löytää GAD-tiimin käyttötarkoituksiin parhaiten soveltuva alustariippumaton kehitystyökalu. Ensin toteutustekniikoihin ja kehitystyökaluihin tutustutaan kirjallisuuskatsauksen avulla. Sen perusteella valitaan viisi mielenkiintoisinta työkalua, joita vertaillaan niille asetettujen minimivaatimusten perusteella. Näistä vii-

destä työkalusta valitaan vertailun perusteella kolme soveltuvinta, joiden avulla toteutetaan esimerkkisovellus Android- ja iOS-kohdealustoille. Lopuksi työkaluja ja niillä toteutettuja sovelluksia vertaillaan mitattavien ja subjektiivisesti arvioitavien kriteerien avulla. Vertailun perusteella tehdään päätös potentiaalisesti käyttöön otettavasta kehitystyökalusta.

Tutkimus etenee teoreettisesta osasta lopussa käsiteltäviin työn tuloksiin ja niiden analysointiin. Luvuissa 2 ja 3 selvitetään mobiilisovelluskehitykseen ja alustariippumattomiin kehitystyökaluihin liittyvää teoreettista taustaa. Luvussa 4 valitaan kolme tässä tutkimuksessa testattavaa työkalua, jotka esitellään tarkemmin luvussa 5. Luvussa 6 käsitellään tutkimuksen tuloksia sekä esitetään tutkimuksen aikana mieleen tulleita jatkotutkimusehdotuksia. Tutkimuksen anti ja suositukset jatkotoimenpiteille esitellään luvussa 7.

2. SOVELLUSKEHITYS MOBIILILAITTEILLE

2.1. Mobiililaitteet ja -sovellukset

Mobiililaitteilla tarkoitetaan tässä älypuhelimia ja tabletteja. Älypuhelimet ovat kannettavia elektronisia laitteita, jotka sisältävät tavallisen matkapuhelimen, kämmentietokoneen ja muiden tietolaitteiden kuten musiikkisoittimen toiminnallisuuksia. Tavallisiin matkapuhelimiin verrattuna älypuhelimille ominaista ovat itse asennettavat ja muokattavat sovellukset, jotka voivat olla myös kolmannen osapuolen (*engl. third party*), kuten käyttäjän itse, rakentamia. (Singh et al. 2008.) Muita älypuhelimelle tunnusomaisia piirteitä ovat kosketusnäyttö, verkkoselain, kattavat verkkoliitännät sekä laiteominaisuudet kuten GPS-paikannin ja kamera. Sovellusten ja toimintojen vuorovaikutuksesta vastaa mobiililaitteella ajettava mobiilikäyttöjärjestelmä.

Tabletin tunnusmerkkinä pidettiin pitkään älypuhelinta suurempaa näyttöä. Nykyään älypuhelinten näyttöjen suuretessa ja tablettien näyttöjen pienetessä raja on kuitenkin hämärtynyt, koska suurimmat älypuhelimet ovat jo pienimpien tablettien kokoluokkaa. Älypuhelimien puhelintoiminnallisuuksia ei tableteista kuitenkaan vielä löydy.

2.2. Mobiilikäyttöjärjestelmät

2.2.1. Markkinatilanne

Vuoden 2013 alussa markkinoita dominoi kaksi mobiilikäyttöjärjestelmää: Googlen Android ja Applen iOS. Strategy Analyticsin laatiman raportin (Mawston 2013) mukaan vuoden 2012 viimeisellä neljänneksellä Android- ja iOS-käyttöjärjestelmät hallitsivat suvereenisti maailmanlaajuisia älypuhelinmarkkinoita arviolta yhteensä 92 prosentin markkinaosuudella. Arvio Androidin maailmanlaajuiseksi markkinaosuudeksi oli 70 prosenttia ja iOS:n 22 prosenttia. Muut alustat vastasivat siis yhteensä ainoastaan kahdeksan prosentin markkinaosuudesta.

Tablettimarkkinoilla oli kahden dominoivan alustan kesken tasaisempaa. IDC:n laatiman raportin (Mainelli et al. 2013) mukaan Android oli jo lähes saavuttanut markkinoita aiemmin ylivoimaisesti hallinneen iOS:n markkinaosuudessa. Erityisesti alle 8-tuumaisten alhaisen hintaluokan tablettien hyvä menekki on vauhdittanut Androidin markkinaosuuden kasvua iOS:än nähden. Niiden osuus oli viime vuoden lopulla jo puolet myydyistä tableteista. iOS:n markkinaosuus oli vuonna 2012 51,3 prosenttia ja Androidin 46,3 prosenttia jättäen muille kilpailijoille ainoastaan 2,4 prosentin osuuden. IDC

ennustaa Androidin ohittavan iOS:n markkinaosuudessa vuoden 2013 aikana, mutta odottaa näiden kahden alustan säilyttävän hallitsevan aseman markkinoilla pidemmällä aikavälillä.

IDC:n ennusteiden (Llamas et al. 2012) mukaan Microsoftin Windows Phone -mobiilikäyttöjärjestelmän odotetaan kasvavan Androidin ja iOS:n kovimmaksi haastajaksi lähivuosien aikana. Vuonna 2013 Windows Phone -käyttöjärjestelmän odotetaan taistelevan älypuhelinmarkkinoiden kolmannesta sijasta ensisijaisesti Research in Motionin uuden BlackBerry 10 -käyttöjärjestelmän kanssa. Seuraavina vuosina Windows Phonen ennustetaan vievän tässä kamppailussa voiton useamman suuren laitevalmistajan, kuten Nokian ja HTC:n, tuen ansiosta. (Llamas et al. 2012.)

Seuraavaksi tutustutaan tarkemmin Android-, iOS- ja Windows Phone -käyttöjärjestelmiin.

2.2.2. Android

Android on Linux-ytimen päälle rakennettu avoimen lähdekoodin käyttöjärjestelmä ja sovelluskokoelma mobiililaitteille. Sen ensimmäinen versio julkaistiin vuonna 2008. Androidin kehittämisestä vastaa Googlen johtamana yli 80 laitevalmistajasta, ohjelmistoyrityksestä ja teleoperaattorista koostuva Open Handset Alliance. (Open Handset Alliance.)

Androidin suurin vahvuus erityisesti iOS:ään nähden on avoimen lähdekoodin mahdollistama muokattavuus. Sama vahvuus on kuitenkin kääntynyt myös sen suurimmaksi heikkoudeksi. Lukuisat laitevalmistajat, usein yhteistyössä teleoperaattoreiden kanssa, ovat julkaisseet omia räätälöityjä versioita käyttöjärjestelmästä omissa laitteissaan. Tämä on johtanut siihen, että tuoreimpien käyttöjärjestelmäpäivitysten toimittaminen on jäänyt räätälöidyn version tehneen laitevalmistajan tai teleoperaattorin vastuulle. Tämän vuoksi päivityksien julkaisu usein pitkittyy, tai niitä ei julkaista lainkaan joillekin laitteille, vaikka laitteisto-ominaisuudet sen sallisivatkin. (McWherter & Gowell 2012, s. 152.) Tästä syystä vanhemmat käyttöjärjestelmäversiot ovat vielä hyvin yleisesti käytössä. Taulukossa 2.1 on esitetty Androidin eri käyttöjärjestelmäversioiden jakauma Google Play -sovelluskaupan käyttäjien laitteissa. Tiedot on kerätty 14 päivän mittaiselta 1.5.2013 päättyneeltä ajanjaksolta (Android Developer).

Kuten taulukosta 2.1 voidaan huomata koodinimillä Froyo ja Gingerbread tunnetut varhaisemmat Android-versiot 2.2 ja 2.3 ovat vielä erittäin suosittuja yli 40 prosentin osuudella kaikista Android-laitteista. Koodinimeä Jelly Bean kantava tuorein käyttöjärjestelmäversio 4.2 on taas otettu käyttöön vasta reilussa kahdessa prosentissa laitteista. Sovelluskehittäjän onkin huomioitava, että saavuttaakseen valtaosan Android-käyttäjistä on kehitettävä sellaisia sovelluksia, jotka toimivat myös huomattavasti vanhemmilla käyttöjärjestelmäversioilla kuin tuorein versio. Tukeakseen valitsemaansa käyttöjärjes-

telmäversiota kehittäjän on käytettävä sitä vastaavaa ohjelmointirajapintaversiota (*engl. application programming interface (API)*).

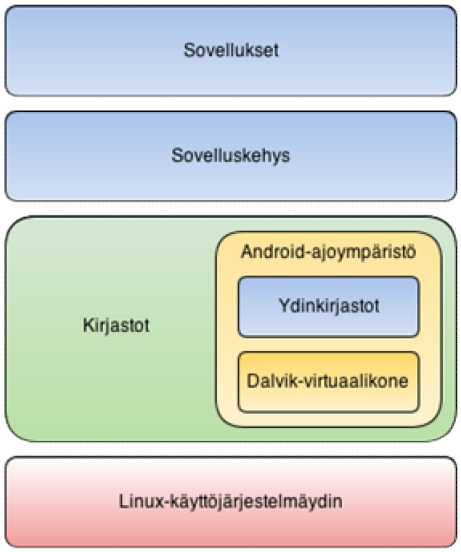
Taulukko 2.1: *Android-käyttöjärjestelmäversioiden osuus Google Play -käyttäjien laitteissa (Android Developer).*

Versio	Koodinimi	API	Jakauma
1.6	Donut	4	0.1%
2.1	Eclair	7	1.7%
2.2	Froyo	8	3.7%
2.3 - 2.3.2	Gingerbread	9	0.1%
2.3.3 - 2.3.7		10	38.4%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sand-wich	15	27.5%
4.1.x	Jelly Bean	16	26.1%
4.2.x		17	2.3%

Ohjelmiston lisäksi myös Android-laitteiden komponenteissa voi olla huomattavia eroavaisuuksia eri valmistajien välillä. Android-laitteita on tarjolla useissa eri näyttökoissa ja -tarkkuuksissa niin älypuhelin- kuin tabletmarkkinoilla (Android Developer). Näyttöjen lisäksi laitteiden suorituskykyyn vaikuttavat komponentit, kuten suoritin ja keskusmuisti, saattavat erota eri hintaluokan laitteissa merkittävästi. Nämä seikat lisäävät sovelluskehittäjän haasteita.

Android-käyttöjärjestelmän arkkitehtuuri on viisikerroksinen (kuva 2.1). Ylin Sovellukset-kerros sisältää kaikki käyttöjärjestelmän tarjoamat sovellukset kuten selain- ja puhelinsovelluksen. Sitä alemman Sovelluskehys-kerroksen päätehtävänä on tarjota ohjelmointirajapintoja ylemmässä kerroksessa olevien sovellusten käyttöön. Kirjastot-kerros tarjoaa sovelluksille taas ydintoiminnallisuuksia esimerkiksi grafiikan piirtoon ja mediatiedostojen esittämiseen. (Maia et al. 2010.)

Android-ajoympäristö-kerros pitää sisällään ydinkirjastojen lisäksi Dalvik-virtuaalikoneen, jonka päällä Android-sovelluksia ajetaan. Dalvik-virtuaalikone on suunniteltu yksinomaan suoritin- ja muistirajoitteisille mobiililaitteille. Alimman kerroksen Linux-käyttöjärjestelmäydin toimittaa ylemmille kerroksille käyttöjärjestelmän ydinpalveluita kuten muistinhallintaa ja vuoronnusta sekä tarjoaa rajapinnat laitteiden käyttöön. (Maia et al. 2010.)



Kuva 2.1: *Android-käyttöjärjestelmän arkkitehtuuri (mukaillen Maia et al. 2010)*

Android-sovelluksia kehitetään tyypillisesti Java-ohjelmointikielellä Googlen suosittelemalla Eclipse-kehitysympäristöllä. Google tarjoaa Eclipseen asennettavaksi Android Developer Tools (ADT) -lisäosan, joka integroi Android SDK:n Eclipseen. Tällöin Android-sovelluksia voidaan testata sen kautta Android-emulaattorilla tai oikealla laitteella. ADT lisää Eclipseen muun muassa graafisen editorin käyttöliittymän rakentamista varten. Kehitystyökalut ovat saatavilla suureen osaan nykyisistä Windows-, Mac- ja Linux-käyttöjärjestelmäversioista. (Android Developer.)

Android-sovellusten jakaminen on täysin vapaata. Sovellusten pääjakelukanava on Google Play -sovelluskauppa, mutta kehittäjä voi jakaa sovelluksiaan halutessaan esimerkiksi omilta verkkosivuiltaan. (Android Developer.)

2.2.3. iOS

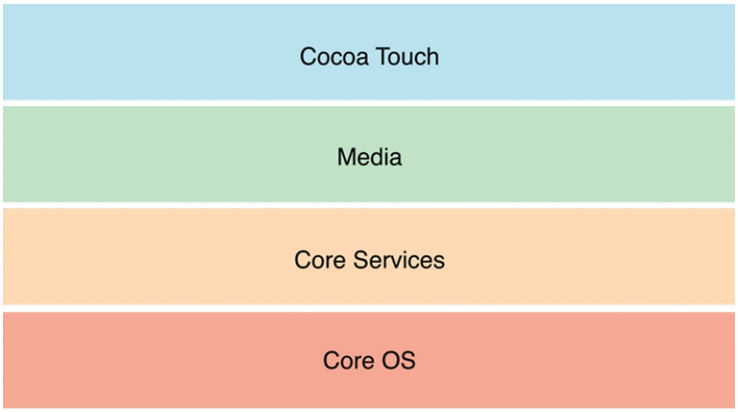
Alun perin vuonna 2007 Applen ensimmäisen älypuhelimen julkaisun yhteydessä iPhone-nimellä julkaistun käyttöjärjestelmän tukea on laajennettu myöhemmin myös muille sen valmistamille mobiililaitteille, jolloin se sai nimekseen iOS. iOS-yhteensopivia mobiililaitteita ovat tällä hetkellä iPhone-älypuhelin, iPad-tabletti ja iPod touch -mediasoitin (Apple iOS).

Apple ei lisensoi iOS-käyttöjärjestelmää käytettäväksi muiden valmistajien laitteilla toisin kuin Google ja Microsoft Android- ja Windows Phone -käyttöjärjestelmiään. Suljetun ekosysteeminsä myötä Apple voi kontrolloida, mitkä sen laitteista saavat tuoreimmat käyttöjärjestelmäpäivitykset ja milloin. Vuoden 2012 syksyllä julkaistu iOS 6 -käyttöjärjestelmäversio tukee iPhone-tuoteperheestä vuonna 2009 julkaistua iPhone 3GS -älypuhelinta, iPad-tuoteperheestä vuonna 2011 julkaistua toisen sukupolven iPadia ja iPod Touch -tuoteperheestä vuonna 2010 julkaistua neljännen sukupolven iPod Touchia sekä kaikkia näiden jälkeen julkaistuja iOS-laitteita (Apple Press Releases).

Applen laitetuki käyttöjärjestelmäpäivityksille ulottuu siis etenkin iPhone-tuoteperheen osalta Android-laitteisiin nähden verrattain pitkälle.

Apple on jakanut käyttöjärjestelmäpäivityksiä iOS 5.0.1 -versiosta lähtien OTA-päivityksinä (*engl. over the air*), jolloin käyttäjän ei tarvitse liittää laitetta kiinni tietokoneeseen päivityksen asennusta varten, vaan päivitys toimitetaan suoraan laitteelle langattoman verkkoyhteyden yli (Friedman 2011). Tämä on nopeuttanut iOS-laitekannan siirtymistä uusimpaan käyttöjärjestelmäversioon. Apple ei julkaise virallisia tilastoja käyttöjärjestelmäversioiden jakaumasta, mutta esimerkiksi suositun Audiobooks-sovelluksen käyttäjistä kerätyn tilaston (iOS Version Stats) perusteella sitä voidaan arvioida. Toukokuun 2013 alussa 89,2 prosentilla kyseisen sovelluksen käyttäjistä oli laitteellaan iOS 6 ja 9,4 prosentilla iOS 5 -versio. Tilastoissa tulee kuitenkin ottaa huomioon, että sovelluksen alin tukema iOS-versio oli 4.3.0. Tuesta sitä edeltäville versioille oli päätetty luopua jo aiemmin merkittävästi vähentyneen käyttäjäosuuden vuoksi.

iOS-käyttöjärjestelmän arkkitehtuuri koostuu neljästä kerroksesta (kuva 2.2). Ylin Cocoa Touch -kerros sisältää tarvittavat kehykset iOS-sovellusten rakentamista varten. Sitä alempana oleva Media-kerros tarjoaa grafiikka-, audio- ja videokirjastoja sovellusten käyttöön, kun taas Core Services -kerros koostuu sovellusten käyttämistä keskeisistä järjestelmäpalveluista. Alin Core OS -kerros sisältää matalan tason järjestelmätoimintoja, joita useimmat ylempien kerrosten kehykset käyttävät. (iOS Technology Overview.)



Kuva 2.2: *iOS-käyttöjärjestelmän arkkitehtuuri (iOS Technology Overview)*

iOS-sovelluksia kehitetään Objective C -ohjelmointikielellä Applen tarjoaman iOS SDK:n avulla. iOS SDK sisältää Xcode IDE:n lisäksi muita siihen integroituvia työkaluja kuten iPhone- ja iPad-simulaattorit. Xcode-kehitystyökalu on saatavilla Mac OS X -käyttöjärjestelmän tuoreimmille versioille. (iOS Developer Program.)

App Store -sovelluskauppa on kuluttajille suunnattujen iOS-sovellusten virallinen ja ainoa jakelukanava. Apple tarjoaa liiketoimintasovelluksille omat jakelumallinsa yritysten sisäiseen ja yritysten väliseen vapaampaan jakeluun. (iOS Developer Program.)

2.2.4. Windows Phone

Windows Phone on Microsoftin kehittämä mobiilikäyttöjärjestelmä, joka korvasi vuonna 2010 Microsoftin aiemman yrityskäyttöön suunnatun Windows Mobile -käyttöjärjestelmän. Vaikka uuden Windows Phone 7 -käyttöjärjestelmän ydin oli edelleen Windows CE -pohjainen, sen Metro UI:ksi kutsuttu käyttöliittymä oli suunniteltu alusta alkaen uudelleen, jotta se vetoaisi paremmin kuluttajamarkkinoihin. Metro UI:n (*nyk. Modern UI*) aloitusnäkymä perustuu dynaamisesti päivittyviin ruutuihin (*engl. live tiles*), jotka näyttävät ajantasaista tietoa kyseiseen sovellukseen liittyvistä tapahtumista, esimerkiksi kalenterisovelluksen ruutu näyttää tietoa tulevista kalenterimerkinnöistä. Tämä eroaa kilpailevien Android- ja iOS-alustojen keskenään samankaltaisista ikonipohjaisista aloitusnäkymistä. (Kokkonen 2012.)

Tuoreimman Windows Phone 8 -käyttöjärjestelmän myötä käyttöjärjestelmän ydin vaihtui Windows NT -pohjaiseksi. Se on sama kuin työpöytäkäyttöön tarkoitetussa Windows 8 -käyttöjärjestelmässä, mikä helpottaa sovellusten kääntämistä työpöytä- ja mobiilialustojen välillä. Windows Phone 8 uudisti myös aiemmin hyvin rajattuja rautavaatimuksia lisäämällä tuen muun muassa moniydinsuorittimille ja SD-muistikorteille. (Kokkonen 2012.) Microsoft ilmoitti ennen Windows Phone 8 -julkaisua, että Windows Phone 7 -älypuhelimet eivät saa Windows Phone 8 -päivitystä, minkä syynä oli todennäköisesti muuttuneet rautavaatimukset (Trew 2012).

Vaikka Microsoft on hallinnut käyttöjärjestelmämarkkinoita työpöytäalustalla jo vuosikymmeniä, se ei ole onnistunut saavuttamaan lähellekään samanlaista suosiota mobiilikäyttöjärjestelmien puolella. Luultavasti suurin syy tähän on myöhäinen painotus kuluttajamarkkinoille, mikä antoi nykyisille markkinajohtajille Applen iOS:lle ja Googlen Androidille vuosien etumatkan Windows Phone -alustaan nähden. Windows Phone -alustalla on kirittävää erityisesti sovelluskaupassa tarjottavien sovellusten määrässä ja laadussa. Tätä varten Microsoftin on houkuteltava käyttäjien lisäksi myös sovelluskehittäjiä puolelleen.

Windows Phone -sovelluksia kehitetään .NET-kehyksellä C#-ohjelmointikielellä Microsoftin tarjoaman Windows Phone SDK:n avulla. Windows Phone SDK sisältää Visual Studio Express IDE:n lisäksi muun muassa Windows Phone -emulaattorin. Windows Phone 8 -sovellusten kehitys vaatii 64-bittisen Windows 8 -käyttöjärjestelmän, kun taas Windows Phone 7 -sovelluskehityksen vaatimuksena on vähintään 32-bittinen Windows Vista tai Windows 7 -käyttöjärjestelmä. (WP Developer; McWherter & Gowell 2012, s. 235.)

Ainoa virallinen jakelukanava kuluttajille suunnatuille Windows Phone -sovelluksille on Windows Phone Store. Microsoft tarjoaa Windows Phone 8 -sovelluksille myös oman yrityksen sisäisen jakelumallinsa. (WP Developer.)

2.2.5. Muut alustat

Markkinoille on tulossa vuoden 2013 aikana useita uusia haastajia kuten Intelin ja Samsungin yhdessä kehittämä Tizen ja suomalaisen Jollan kehittämä Sailfish, jotka molemmat ovat jatkaneet kehitystä Nokian hylkäämän MeeGo-käyttöjärjestelmä pohjalta. Firefox-selaimistaan tunnettu Mozilla aikoo myös ottaa osaa mobiilialustojen väliseen kilpaan julkaisemalla oman Firefox OS -käyttöjärjestelmän tänä vuonna. (Olson 2013.)

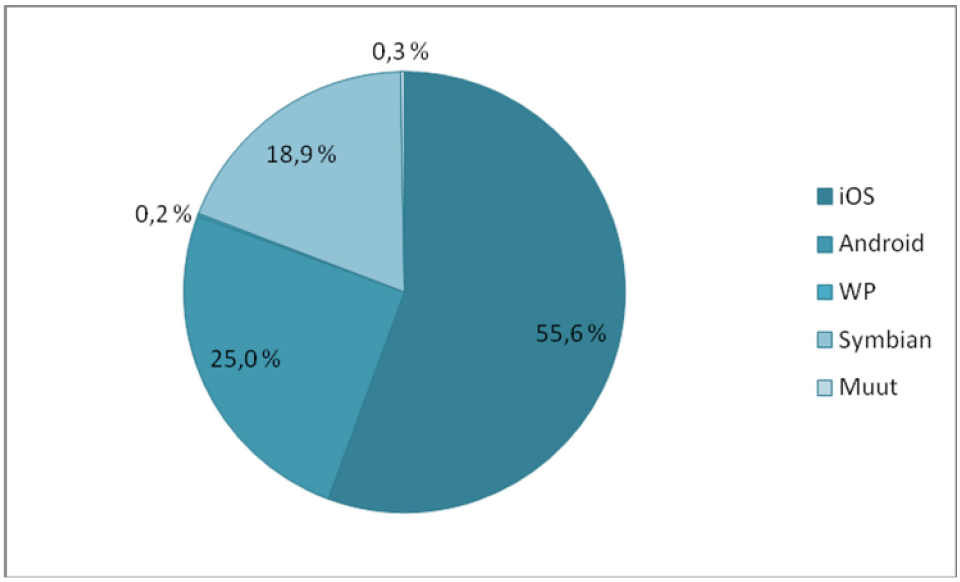
Edellä mainittuja käyttöjärjestelmiä yhdistää se, että ne pohjautuvat Linux-ytimeen ja avoimeen lähdekoodiin. Niiden tähtäimessä ovat erityisesti alemman hintaluokan älypuhelinmarkkinat. Suurimpana haasteena uusilla tulokkailla on kuroa kiinni eroa sovellustarjonnassa markkinoita dominoiviin alustoihin nähden. Tizen ja Firefox OS luottavat erityisesti HTML5-pohjaisiin web-sovelluksiin, joiden kehittäminen on nopeampaa ja edullisempaa kuin natiivisovellusten kehittäminen, sillä web-sovelluksille on jo olemassa laaja kehittäjäkunta. (Brown 2012.)

Uusien haastajien lisäksi markkinoilla on vielä väistyviä mobiilikäyttöjärjestelmiä, joista merkittävin on Symbian, Nokian ensisijainen älypuhelinkäyttöjärjestelmä ennen Windows Phonea. Nykyisissä myyntitilastoissa mitattuna Symbian-älypuhelimilla on hyvin pieni osuus markkinoista, mutta vuosien takaisen menestyksensä vuoksi sen laitekanta on edelleen merkittävä. Symbian-älypuhelinten toimitusten oli kuitenkin määrä loppua vuoden 2013 kesällä, mikä sinetöi alustan lopullisen kohtalon (Thomas 2013).

2.3. Mobiililaitteet Metsolla

Metson mobiililaitekannasta saa ainakin suuntaa antavan kuvan tutustumalla IBM Notes Traveler -sovelluksen käyttäjätilastoihin. IBM Notes on Metsossa käytössä oleva sähköposti- ja työryhmäohjelmisto, jonka sähköpostin, kalenterin ja yhteystiedot Notes Traveler -sovellus tarjoaa mobiililaitteille. Metson sisäverkossa jaeltava Notes Traveler -versio tukee Android ja iOS -käyttöjärjestelmien älypuhelimia ja tabletteja sekä Windows Phone ja Symbian -käyttöjärjestelmien älypuhelimia (Metso Avenue). Koko Metson laajuisesti Notes Traveler on käytössä lähes 10 000 mobiililaitteessa (Metso Notes Traveler DB).

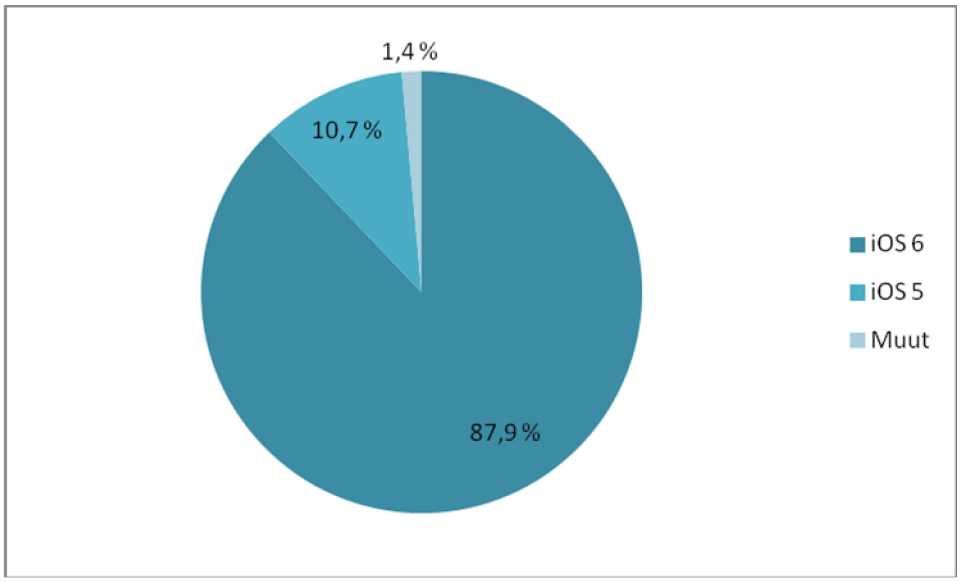
Kuvassa 2.3 on esitetty mobiilikäyttöjärjestelmien jakauma Notes Travelerin MAC-käyttäjien mobiililaitteissa toukokuun alussa 2013. Metso MAC-segmentin sisällä Notes Traveleria käytti tuolloin yhteensä 2739 mobiililaiteitta. (Metso Notes Traveler DB.) Tässä tutkimuksessa kehitetyt mobiilisovellukset ovat suunnattu juuri näille laitteille.



Kuva 2.3: *Mobiilikäyttöjärjestelmien jakauma Metso MAC:n Notes Traveler -käyttäjien mobiililaitteissa toukokuun alussa 2013*

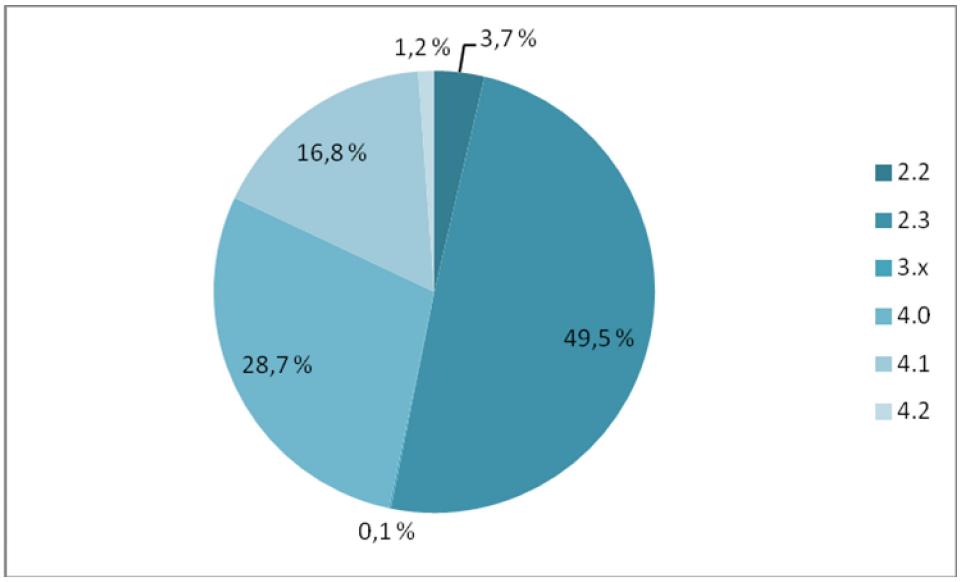
Kuten kuvasta 2.3 voidaan todeta, iOS oli selvästi suosituin mobiilikäyttöjärjestelmä MAC-segmentin mobiilikäyttäjien keskuudessa yli 55 prosentin osuudella. Android oli toiseksi suosituin 25 prosentin osuudella mobiililaitteista. Kolmanneksi suosituimman Symbianin osuus (18,9 %) oli verrattain suuri, mutta sen voidaan odottaa laskevan laitekannan uusiutuessa. Windows Phone sai Notes Traveler -tuen vasta huhtikuun alussa, mikä selittänee osaltaan sen marginaalista osuutta (0,2 %) (Metso Avenue). Windows Phone -laitemäärän voidaan kuitenkin odottaa kasvavan tulevaisuudessa hyvän Microsoft Office- ja Sharepoint-tuen vuoksi.

Androidin ja iOS:n hallinta näyttäisi olevan ajankohtaista kuluttajamarkkinoiden lisäksi myös Metson MAC-segmentin mobiililaitekannassa. Metso-omisteisten puhelinten valinta tehdään alueellisen Metso IT:n laatimien suositusten mukaan. Tämän vuoksi erityisesti iPhone-mallit ovat olleet suosittuja, koska ne löytyvät lähes jokaisen alueen suosituslistalta (Metso Avenue). Android-mallit ovat taas saaneet erityistä suosiota alueilla, joissa puhelimet ovat osittain omakustanteisia iPhonea matalamman hinnan vuoksi. Kuvassa 2.4 on esitetty iOS- ja kuvassa 2.5 Android-käyttöjärjestelmäversioiden jakaumat MAC-käyttäjien mobiililaitteissa.



Kuva 2.4: *iOS-käyttöjärjestelmän versiojakauma Metso MAC:n Notes Traveler -käyttäjien mobiililaitteissa toukokuun alussa 2013*

Lähes 90 prosenttia MAC-segmentin iOS-laitteiden haltijoista oli päivittänyt laitteen käyttöjärjestelmän vuoden 2012 syksyllä julkaistuun tuoreimpaan iOS 6 -versioon (kuva 2.5). Toiseksi tuorein iOS 5 -versio oli käytössä reilussa 10 prosentissa laitteista ja sitä vanhempia käyttöjärjestelmäversioita oli käytössä vain reilun prosentin verran. Tämä vastaa myös arvioita kuluttajamarkkinoilla vallitsevasta tilanteesta.



Kuva 2.5: *Android-käyttöjärjestelmän versiojakauma Metso MAC:n Notes Traveler -käyttäjien mobiililaitteissa toukokuun alussa 2013*

Android-laitekannassa käyttöjärjestelmän versiojakauma oli yhtä sirpaloitunut kuin kuluttajamarkkinoilla (kuva 2.5). Merkittävin huomio oli, että lähes puolet (49,5 %) Android-laitteista käytti vielä vanhaa 2.3-käyttöjärjestelmäversiota. Yhdessä sitä vanhemman 2.2-version kanssa Android 2.x -laitteita oli yli puolet Metso MAC:n

Android-laitteista, kun taas tuoreimmalla 4.x-versiolla käyttäjiä oli vajaaksi jäänyt toinen puolikas laitekannasta. Tableteille suunnatun 3.x-käyttöjärjestelmäversion osuus (0,1 %) oli täysin marginaalinen.

3. ALUSTARIIPPUMATON MOBIILISOVELLUS-KEHITYS

3.1. Tarve alustariippumattomalle mobiilisovelluskehitykselle

Mobiililaitteiden laitteiston ja ohjelmiston nopea kehitys viimeisen kymmenen vuoden aikana on johtanut siihen, että yksinkertaisista taskuun mahtuvista kommunikointivälineistä on tullut täysiverisiä tietokoneita itse asennettavine sovelluksineen ja verkkoselaimineen. Mikkosen ja Taivalsaaren (2011) mukaan verkkoresursseja hyödyntävien työpöytä- ja mobiilisovellusten kehitys on lähtenyt kuitenkin viime vuosien aikana eri suuntiin. Työpöytäkäytössä verkkoselain on yleisin sovellus, jolla käyttäjät hyödyntävät verkossa olevia resursseja, kun taas mobiililaitteissa verkkoresursseja käytetään yleisemmin kyseisiä käyttötapauksia varten räätälöityjen erillisten natiivisovellusten avulla.

Mikkonen ja Taivalsaari (2011) esittävät tähän syyksi toisistaan olennaisesti poikkeavat käyttötavat erilaisilla päätelaitteilla. Koska mobiililaitteita käytetään lähes missä ja milloin tahansa, käyttäjät haluavat suorittaa toimintoja niillä nopeammin ja vaivattomammin kuin työpöytäkäytössä. Esimerkiksi päivän sään tarkastaminen kävellessä älypuhelimen pieneltä näytöltä on sitä miellyttävämpää mitä vähemmän aikaa ja painalluksia toiminnon suorittamiseen tarvitaan. Näitä seikkoja tavanomaiset työpöytäkäyttöön suunnitellut verkkosivut eivät ole huomioineet, joten parhaan käyttökokemuksen ovat tähän saakka tarjonneet toimintoja varten räätälöidyt natiivisovellukset.

Natiivisovelluksella tarkoitetaan tässä laitteelle asennettavaa ohjelmaa, joka on tavallisesti rakennettu kohteena olevan mobiilialustan kehittäjän tarjoamien kehitystyökalujen, -ohjeiden sekä alustaa tukevien ohjelmointikielten avulla. Esimerkiksi iPhone-sovellusta, joka on kehitetty iOS SDK:n avulla Objective C -ohjelmointikielellä Applen tarjoamien suunnittelu- ja tyyliohjeiden mukaisesti, kutsutaan natiivisovellukseksi. Natiivisovellusten vahvuutena pidetään erityisesti yhtenäistä käyttötuntumaa käyttöjärjestelmän sovelluksiin nähden, koska niiden käyttöliittymä rakennetaan käyttäen samoja käyttöliittymäkirjastoja kuin mitä käyttöjärjestelmän omissa sovelluksissa on hyödynnetty. Käyttöjärjestelmän kehittäjän jakelemat kirjastot takaavat myös ajantasaisimmat rajapinnat käyttöjärjestelmän tukemiin laitekohtaisiin ominaisuuksiin kuten GPS-paikannukseen ja kameraan.

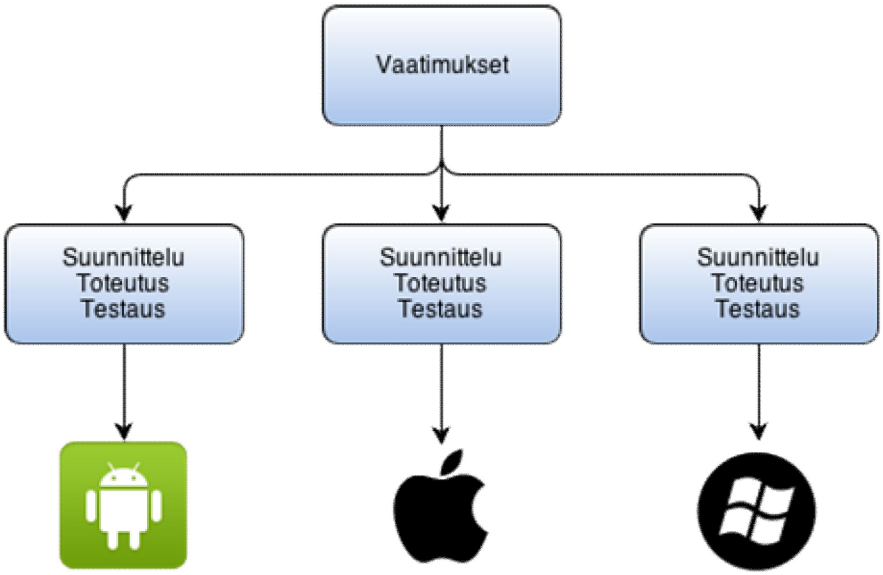
Mikäli ohjelmistokehittäjä haluaa tehdä natiivisovelluksen useammalle mobiilikäyttöjärjestelmälle, hänen täytyy hallita eri käyttöjärjestelmille ominaisten kehitystyökalujen, ohjelmointikielten ja -rajapintojen lisäksi käyttöjärjestelmäkohtaiset sovellusten suunnittelu- ja tyyliperiaatteet. Esimerkiksi toteuttaakseen natiivisovelluksen Androidille ja iOS:lle on kehittäjän osattava Java ja Objective C -ohjelmointikielten lisäksi työskennellä molemmille mobiilialustoille ominaisten kehitystyökalujen, kuten Eclipse- ja Xcode-ohjelmointiympäristöjen sekä niihin integroituvien analysointi- ja testaustyökalujen, parissa. Tämän lisäksi kehittäjän on tunnettava käyttöjärjestelmäkohtaisia sovelluksen tyyli- ja suunnitteluperiaatteita muun muassa erilaisten käyttöliittymäelementtien käyttöön ja sovelluksen eri tiloihin liittyen.

Kehitys monimutkaistuu entisestään, kun sovelluksen on tuettava useampia eri muunnelmia ja versioita eri käyttöjärjestelmistä. Tämä johtuu siitä, että käyttöjärjestelmien päivitykset tuovat useimmiten uusia ominaisuuksia, joita vanhat käyttöjärjestelmäversiot eivät tue. Tällöin uudelle käyttöjärjestelmäversiolle rakennettu sovellus ei enää toimi sitä vanhemmilla versioilla käyttöjärjestelmästä. Kuten jo aiemmin kohdassa 2.2 esitettiin erityisesti Android-käyttöjärjestelmän kannalta tuki useammille eri versioille ja muunnelmille on tärkeää. Android on hyvin sirpaloitunut mobiilialusta otettaessa huomioon sekä sen eri versioiden käyttöaste että useiden laitevalmistajien siihen tekemät omat muunnelmat.

Wassermanin (2010) mukaan ohjelmistokehittäjällä on neljä eri lähestymistapaa toteuttaa mobiilisovelluksia, jotka tukevat useita mobiilialustoja ja niiden eri muunnelmia:

1. Kehittää natiivisovellus yhdelle mobiilialustalle kerrallaan hyödyntäen mahdollisimman paljon alustan eri muunnelmille yhteisiä kirjastoja, jolloin sama sovellus toimii varmemmin saman käyttöjärjestelmän eri versioilla.
2. Kehittää natiivisovellus jokaiselle mobiilialustalle ja niiden muunnelmalle erikseen.
3. Kehittää selaimella ajettava mobiililaitteille suunnattu web-sovellus.
4. Kehittää (alustariippumattomilla kehitystyökaluilla) abstraktiokerroksien avulla samaa koodikantaa useilla mobiilialustoilla hyödyntävä natiivisovellusten tapaan ajettava sovellus.

Näistä Wassermanin esittämä toinen lähestymistapa vaatii eniten resursseja. Usean mobiilialustan tuen lisäksi tuki usealle eri alustan muunnelmalle tarkoittaa käytännössä yhtä usean eri sovellusversion kehitystä ja ylläpitoa. Ohjelmistokehittäjä pääsee huomattavasti helpommalla käyttäen ensimmäistä lähestymistapaa, mutta menettää samalla mahdollisuuden hyödyntää alustan uusien versioiden myötä tulleita ominaisuuksia. Natiivisovellusten kehitys usealle tällä hetkellä valta-asemassa olevalle mobiilialustalle, kuten Androidille, iOS:lle ja Windows Phone:lle, vaatii joka tapauksessa kohdealustojen määrään nähden yhtä usean ohjelmistoprojektin läpikäymistä (kuva 3.1).



Kuva 3.1: *Natiivisovellusten kehitys usealle kohdealustalle*

Seuraavissa kohdissa 3.2 ja 3.3 käydään läpi Wassermanin (2010) esittämät kolmas ja neljäs lähestymistapa alustariippumattomien mobiilisovellusten kehittämiseen.

3.2. Web-sovellukset

Web-sovellusten etu natiivisovelluksiin nähden on, että niitä ei tarvitse natiivisovellusten tavoin asentaa laitteelle, vaan niitä käytetään laitteen selaimella URL:n eli web-osoitteen kautta. Sovelluksen toiminta ei ole tällöin riippuvainen käyttöjärjestelmästä, vaan selaimen tukemista ominaisuuksista. Samaa web-sovellusta voidaan täten käyttää usealla eri mobiilialustalla, kunhan sovellusta toteutettaessa selainten yhteensopivuuksiin liittyvät seikat on huomioitu. Suunniteltaessa web-sovelluksia mobiililaitteille on tämän lisäksi kiinnitettävä erityistä huomiota vaihteleviin näyttökokoihin sekä erilaiseen käyttöfilosofiaan verrattaessa työpöytäkäyttöä varten suunniteltaviin web-sovelluksiin (Mikkonen & Taivalsaari 2011).

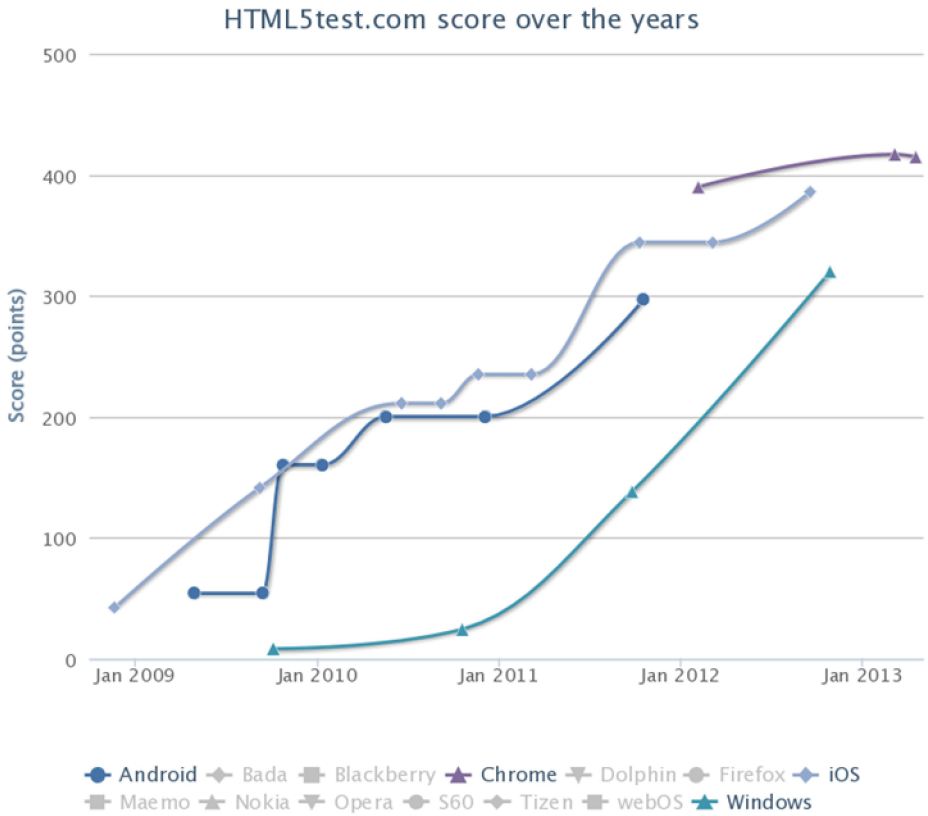
Web-sovellusten toinen merkittävä etu verrattaessa natiivisovelluksiin on sovelluspäivitysten jakelun vaivattomuus. Sovelluspäivitykset asennetaan ainoastaan web-sovelluksen alustana toimivalle palvelimelle, josta ne ovat tämän jälkeen välittömästi kaikkien käyttäjien saatavilla. Web-sovelluksen päivitysprosessi on käyttäjän näkökulmasta huomaamatonta toisin kuin natiivisovelluksen päivitysprosessi, joka useimmiten vaatii sen, että käyttäjä käy erikseen hyväksymässä päivityksen latauksen sitä tarjoavasta sovelluskaupasta. Tämä mahdollistaa sen, että web-sovellusten päivityksiä voidaan jaella pienissä palasissa jopa useita kertoja päivässä, niin sanottuina nano-päivityksinä, keskeyttämättä sovelluksen toimintaa (Mikkonen & Taivalsaari 2011).

Mobiilien web-sovellusten kehityksessä on avainasemassa vielä kehitteillä oleva HTML5-standardi. W3C:n ja WHATWG:n (*Web Hypertext Application Technology*

Working Group) yhdessä kehittämä HTML5 tuo uudistuneen merkintäkielen lisäksi joukon uusia ohjelmointirajapintoja, joita voidaan hyödyntää web-standardeihin kuuluvien CSS3-kuvauskielen ja JavaScript-ohjelmointikielen avulla. Edeltävään HTML4-standardiin nähden HTML5 tarjoaa uusia ohjelmointirajapintoja muun muassa edistyneemmälle grafiikalle ja käyttöliittymätapahtumille, video- ja äänimediaelementeille sekä verkkoyhteydettömän käytön mahdollistavalle lokaalille tietokannalle (W3C: HTML5 differences from HTML4). Näiden uudistusten ansiosta web-sovellukset sekä näyttävät että käyttäytyvät yhä enemmän kuten laitteelle asennettavat natiivisovellukset.

Kaikkiin käyttötarkoituksiin web-sovellukset eivät kuitenkaan sovi. Ohrtin ja Turaun (2012, s. 73) mukaan käyttöjärjestelmään integroitumattomilla sovelluksilla (*engl. non-integrated apps*), kuten selaimella käytettävillä web-sovelluksilla, ei ole pääsyä kaikkiin laiteominaisuuksiin toisin kuin laitteelle asennettavilla integroituvilla sovelluksilla (*engl. integrated apps*). Esimerkiksi Bluetooth-yhteyden hyödyntäminen selaimen kautta ei ole ainakaan vielä mahdollista. Uusia laiteominaisuuksia tukevia rajapintoja tulee kuitenkin koko ajan lisää uusien HTML5-määritysten myötä.

Eri mobiiliselainten tuki HTML5-määritteille on vaihtelevaa. Erityisesti Windows Phone -käyttöjärjestelmän oletusselaimen, Internet Explorerin, HTML5-tuki on ollut ajoittain hyvin paljon jäljessä kilpailevien alustojen oletusselaimiin nähden (kuva 3.2).



Kuva 3.2: *HTML5test.com-sivuston HTML5-testissä saavutettujen pistemäärien kehitys Android (Android, Chrome), iOS ja Windows Phone -mobiilikäyttöjärjestelmien oletusselaimilla (HTML5test.com).*

Kuvassa 3.2 on esitetty HTML5test.com-sivuston laatiman HTML5-testin pistemäärät Android, iOS ja Windows Phone -käyttöjärjestelmien oletusselaimilla vuodesta 2009 alkaen. Testin tarkoituksena on antaa käsitys siitä, kuinka hyvin kukin selain tukee HTML5:n ominaisuuksia. Testattavia ominaisuuksia ovat sekä W3C:n määritelmän mukaiset HTML5-ominaisuudet että siihen läheisesti liittyvät ominaisuudet kuten 3D-efektit HTML5:n canvas-elementin sisällä mahdollistava WebGL. Kaikkia HTML5-ominaisuuksia testi ei kuitenkaan käy läpi. (HTML5test.com.)

Selain saa testissä pisteitä tukemiensa ominaisuuksien perusteella – mitä lähempänä selaimen ansaitsema pistemäärä on sen hetkistä maksimipistemäärää, sitä kattavampi sen HTML5-tuki on. Tällä hetkellä HTML5-testin maksimipistemäärä on 500, jota lähimmäksi kolmen tutkitun mobiilikäyttöjärjestelmän oletusselaimista pääsi Android 4 -käyttöjärjestelmän Google Chrome 25 -selain. (HTML5test.com.) Yleisesti ottaen tällä hetkellä tuoreimpien käyttöjärjestelmäversioiden oletusselaimet tukevat HTML5-ominaisuuksia varsin hyvin.

Internet Explorerin välimatkaa muihin mobiiliselaimiin selittänee sen käyttämä oma selainmoottori Trident. Sekä iOS:n Safari-selain että Androidin Android- ja Google Chrome -selaimet käyttävät samaa avoimen lähdekoodin WebKit-selainmoottoria, vaikkakin Google aikoo siirtyä käyttämään omaa, WebKitistä haarautunutta, Blink-selainmoottoria seuraavissa Chrome-selainversioissa (Paul 2013). Samaa selainmoottoria käyttävien selainten toiminnalliset erot ovat luonnollisesti pienempiä verrattuna eri selainmoottoria käyttäviin selaimiin. Eroja WebKit-pohjaisten selaimien välillä HTML5-tuessa kuitenkin on. Esimerkiksi Android-käyttöjärjestelmän Android- ja Chrome-selaimet tukevat mediakaappauksia (*engl. media capture*), kuten kuvien ottamista laitteen kameralla tai äänen nauhoittamista laitteen mikrofonilla, kun taas iOS:n Safari-selaimen tuoreimmassa versiossa tuki on vain osittainen. Windows Phonen Internet Explorer ei taas tue ominaisuutta vielä lainkaan. (Mobile HTML5.)

Rajallisen laiteominaisuustuen lisäksi web-sovellusten heikompi suorituskyky on usein syy siihen, miksi niiden käyttäjäkokemus (*engl. user experience*) koetaan natiivisovelluksia heikommaksi. Vaikka JavaScriptin suorituskyky selaimissa on parantunut koko ajan, hidastaa selaimen lisäämä ylimääräinen kerros sovelluksen ja laitteiston välissä web-sovellusten suorituskykyä verrattaessa natiivisovelluksiin. Toisin kuin natiivisovellukset, joiden ohjelmakoodi käännetään useimmiten suoraan laitteistolla ajettavaksi konekieleksi, web-sovellukset käyttävät ajoympäristönään selainta, joka tulkkaa JavaScript-koodia ajonaikaisesti kohdealustalle. Ennen kuin JavaScriptiä voidaan ajaa selaimen muistista, täytyy se ensin ladata sekä jäsentää, mikä hidastaa suorituskykyä entisestään. (Charland 2011; Juntunen et al. 2013.)

3.3. Alustariippumattomat kehitystyökalut

Alustariippumattomilla kehitystyökaluilla (*engl. cross-platform development tools*) tarkoitetaan ohjelmistokehitykseen suunnattuja työkaluja, joilla voidaan rakentaa ja jaella sovelluksia usealle kohdealustalle hyödyntäen mahdollisimman paljon yhteistä ohjelmakoodia tuettavien alustojen välillä (Jones et al. 2012, s. 6). Niiden tarkoituksena on vähentää kehitykseen kuluvaa aikaa ja kustannuksia usealle alustalle kohdistuvassa sovelluskehityksessä. Alustariippumattomat kehitystyökalut ovat kasvattaneet suosiotaan viime vuosien aikana erityisesti alati muuttuvalla, pirstoutuneella mobiilirintamalla. VisionMobilen (Jones et al. 2012) raportin mukaan vuoden 2012 alussa oli saatavilla yli 100 alustariippumatonta kehitystyökalua.

Jones et al. (2012, s. 22) mukaan mobiilialustoille suunnatut alustariippumattomat työkalut voidaan jakaa viiteen eri luokkaan niiden teknologisen lähestymistavan mukaan:

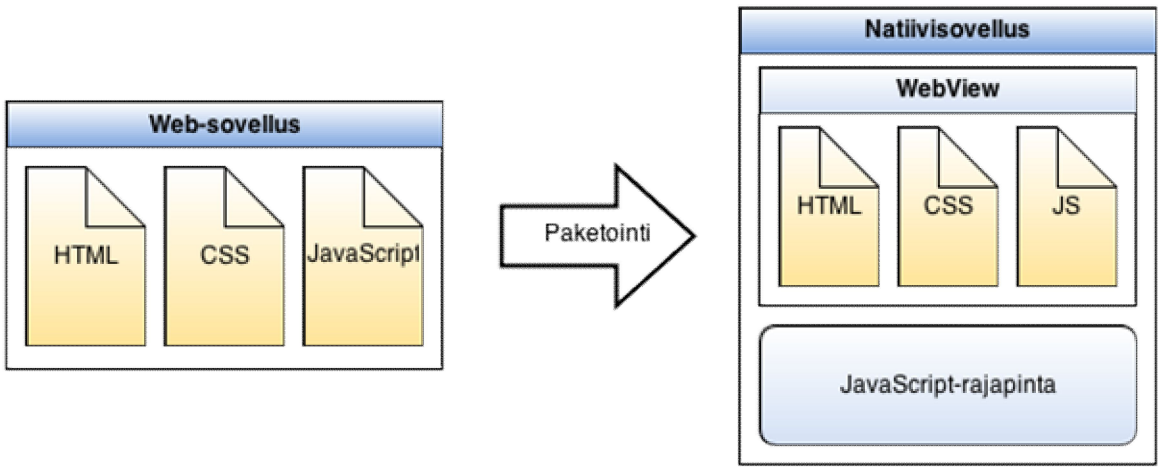
- JavaScript-kehykset
- sovellustehtaat (*engl. app factory*)
- web-natiivi-käärijät (*engl. web-to-native wrapper*)
- ajoympäristöt
- lähdekoodin kääntäjät.

JavaScript-kehyksillä tarkoitetaan tässä koodikirjastoja, joiden avulla pyritään helpottamaan web-tekniikoilla rakennettavien mobiilikäyttöliittymien luomista. Useimmiten tällaiset kirjastot tarjoavat tuen kosketuseleille ja natiivisovellusten kaltaisille käyttöliittymäelementeille. (Jones et al. 2012, s. 22.) Tunnetuin esimerkki mobiilaitteille suunnatuista JavaScript-kehyksistä on jQuery Mobile, joka on rakennettu erittäin suositun jQuery JavaScript-kirjaston päälle (jQueryMobile).

Sovellustehtaat ovat visuaalisia suunnittelutyökaluja, joiden avulla mobiilisovelluksia voidaan kehittää nopeasti ilman ohjelmointitaitoja. Sovelluksen käyttöliittymä rakennetaan useimmiten joko seuraamalla avustajan (*engl. wizard*) ohjeita, muokkaamalla suunnittelutyökalusta löytyviä valmiita pohjia tai koostamalla se itse työkalun tarjoamista valmiista käyttöliittymäelementeistä. Sovellustehtaiden avulla tuotettujen sovellusten toiminnallisuudet ovat useimmiten hyvin rajattuja ja yksinkertaisia koodivapaan lähestymistavan vuoksi. (Jones et al. 2012, s. 22.) Esimerkiksi selaimessa ajettava AppMakr-työkalu on keskittynyt tuottamaan RSS-syötteitä lukevia uutissovelluksia (AppMakr).

Web-natiivi-käärijän avulla HTML5-, CSS- ja JavaScript-tekniikoilla toteutettu websovellus voidaan paketoida ajettavaksi natiivisovelluksen rungon sisällä (kuva 3.3). Tällöin sovelluksesta tulee eräänlainen web-sovelluksen ja natiivisovelluksen välimuoto, **hybridisovellus**. Tekniikka perustuu kaikista mobiilikäyttöjärjestelmistä löytyvään

WebView-komponenttiin. Se on natiivisovelluksen sisällä oleva näkymä, eräänlainen selaimen instanssi, joka kykenee esittämään web-sisältöä. (Jones et al. 2012, s. 30.)

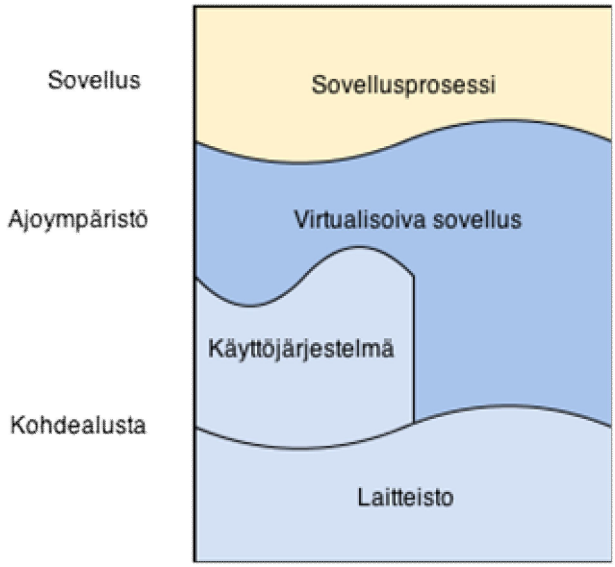


Kuva 3.3: *Web-sovelluksen paketointi natiivisovelluksen rungon sisälle hybridisovellukseksi*

Tavallisiin web-sovelluksiin nähden hybridisovellukset voivat käyttää niitä laajemmin laiteominaisuuksia, kuten kameraa, web-natiivi-käärijän lisäämien JavaScript-rajapintojen avulla. Täysin verkkoyhteydetön käyttö on myös mahdollista, kun tarvittavat tiedostot on paketoitu sovelluksen sisälle, eikä niitä tarvitse ladata ensin palvelimelta kuten web-sovelluksissa. Tunnetuin esimerkki web-natiivi-käärijästä on PhoneGap, jonka avulla web-sovellus voidaan paketoida hybridisovellukseksi Android-, iOS-, Windows Phone-, BlackBerry-, webOS-, Symbian- ja Bada-käyttöjärjestelmille (PhoneGap).

Ohrt ja Turau (2012) käyttävät **ajoympäristön** sisällä ajettavista mobiilisovelluksista nimitystä **tulkattavat sovellukset** (*engl. interpreted applications*). Toisin kuin natiivisovellukset, joiden konekielelle käännetty ohjelmakoodi ajetaan suoraan laitteistolla, tulkattavat sovellukset käyttävät suorituksensa aikana omaa ajoympäristöä, joka muuntaa ohjelmakoodin kohdealustan ymmärtämään muotoon. Ajoympäristöstä Ohrt ja Turau käyttävät yleisnimitystä virtuaalikone riippumatta siitä, onko kyseessä tulkki vai prosessikohtainen virtuaalikone.

Virtualisoinnin avulla sovellusta voidaan ajaa oman hiekkalaatikon (*engl. sandbox*) sisällä ilman, että se on tietoinen kohdealustan yksityiskohdista (kuva 3.4). Kohdealustalle rakennettu virtuaalikone huolehtii yhteydenpidosta ja yhteensopivuudesta sovelluksen ja kohdealustan välillä, kunhan sovellus käyttää virtuaalikoneen tarjoamia standardoituja ohjelmointirajapintoja. Virtuaalikone ei vastaa mitään oikeaa alustaa, minkä ansiosta sen sisällä ajettava sovellus voidaan kirjoittaa korkean tason ohjelmointikielellä omassa kehitysympäristössään ottamatta toteutuksessa kantaa siihen millä kohdealustoilla sen tulisi toimia. (Smith & Nair 2005.)



Kuva 3.4: *Prosessikohtainen virtuaalikone (mukaillen Smith & Nair 2005)*

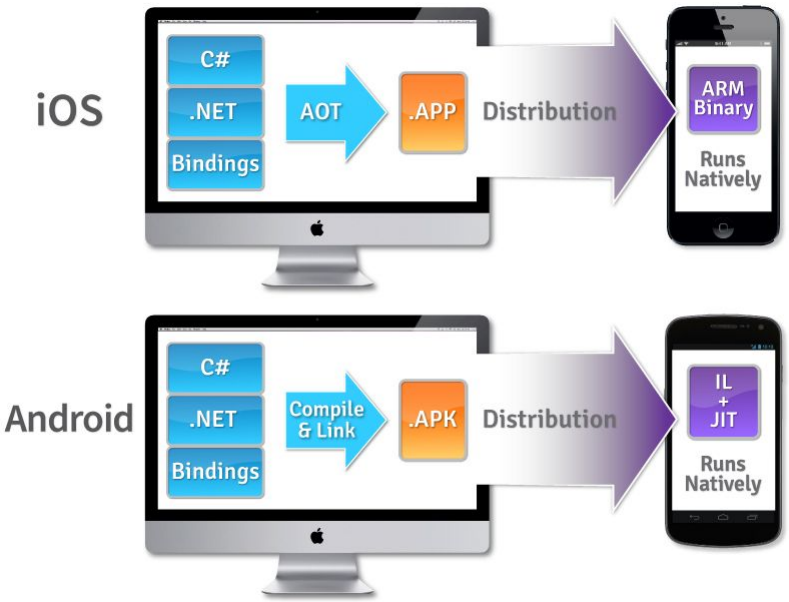
Virtuaalikone sisältää yksinkertaisimmillaan tulkin, joka muuntaa käskyjä ajonaikaisesti käsky kerrallaan kohdealustan ymmärtämään muotoon. Tämä voi olla hyvin hidasta, koska tulkin täytyy tehdä sama muunnos joka kerta, kun kyseisen käskyn luo saavutaan. (Smith & Nair 2005.) Toisaalta tulkkaus nopeuttaa ohjelman kehitystyötä, koska ohjelmakoodia ei tarvitse kääntää joka kerta ennen ohjelman ajoa.

Edistyneemmät prosessikohtaiset virtuaalikoneet osaavat kääntää ohjelmakoodin kohdealustan ymmärtämälle konekielelle, jolloin sovellusta voidaan ajaa suoraan kohdealustalla nopeuttaen ohjelman suoritusta (Smith & Nair 2005). Ohjelmakoodi käännetään tyypillisesti joko ennen ohjelman suoritusta (*engl. ahead-of-time compilation, lyh. AOT*) tai ohjelman suorituksen aikana (*engl. just-in-time compilation, lyh. JIT*) (Jones et al. 2012, s. 24).

Oli kyseessä sitten yksinkertainen tulkki tai edistyneempi prosessikohtaisen virtuaalikone, sen toteuttaminen kohdealustalle on huomattavasti yksinkertaisempaa kuin kokonaisen kääntäjän toteuttaminen. Tämän ansiosta tulkattujen sovellusten siirtäminen alustalta toiselle on helpompaa kuin natiivisovellusten. (Smith & Nair 2005.) Esimerkiksi JavaScipt-ajoympäristöä hyödyntävän Appcelerator Titaniumin avulla voidaan rakentaa sovelluksia Android-, iOS-, BlackBerry- ja Tizen-alustoille (Appcelerator Titanium Docs).

Lähdekoodin kääntäjät muuntavat lähdekoodin joko suoraan konekielelle tai jollekin välitason kielelle (*engl. intermediate language*) kuten tavukoodiksi. Suoraan kohdealustan laitteistolla ajettava konekieli on kaikista nopeinta suorittaa, mutta alustakohtaisten kääntäjien toteuttaminen on monimutkaista. Tämän vuoksi lähdekoodin kääntäjiä käytetään useimmiten yhdessä ajoympäristön kanssa. (Jones et al. 2012, s. 24.) Tällöin kyseessä on tyypillisesti kääntäjä-elementin sisältävä prosessikohtainen virtuaalikone.

Eräs esimerkki kääntäjä- ja ajoympäristö-elementin sisältävästä alustariippumattomasta kehitystyökalusta on Xamarin. Sen avulla voidaan toteuttaa Android- ja iOS-sovelluksia Windows Phone -alustan tukeman .NET-ajoympäristön sisälle C#-ohjelmointikielellä. Lähdekoodin käännösprosessi on erilainen eri alustoilla (kuva 3.5). iOS-sovelluksissa ohjelmakoodi käännetään ennen ohjelman suoritusta staattiseksi binääritiedostoksi, jolloin lopputuloksena on natiivisovellus. Android-sovelluksissa käännösprosessi on kaksiportainen. Kun Android-sovellus paketoidaan, lähdekoodi käännetään ensin välitason kielelle. Sovellusta ajettaessa ajoympäristö huolehtii välitason kielen kääntämisestä kohdealustan ymmärtämään muotoon, jolloin kyseessä on tulkattava sovellus. (Xamarin How It Works.)



Kuva 3.5: *Xamarin-kehitystyökalulla luodun sovelluksen käännösprosessi Android- ja iOS-alustoille (Xamarin How It Works)*

Xamarinin tavoin myös useat muut alustariippumattomat kehitystyökalut hyödyntävät useampaa kuin yhtä edellä mainituista lähestymistavoista. Esimerkiksi Sencha Touch Bundle sisältää JavaScript-kehyksen lisäksi Sencha Cmd -nimisen web-natiivi-käärijä-elementin (Sencha). Työkaluja onkin kenties helpompi luokitella sen mukaan, minkä tyyppisiä sovelluksia niiden avulla voidaan rakentaa.

Ohrt ja Turau (2012) jaottelevat alustariippumattomilla kehitystyökaluilla rakennetut mobiilisovellukset neljään kategoriaan:

- natiivisovelluksiin, jotka käyttävät järjestelmän rajapintoja suoraan
- natiivisovelluksiin, jotka käyttävät järjestelmän rajapintoja abstraktion tarjoavan kirjaston kautta
- tulkattaviin sovelluksiin, jotka sisältävät virtuaalikoneen asennuspaketissaan
- tulkattaviin sovelluksiin, joiden virtuaalikone asennetaan erillisenä sovelluksena.

Tässä työssä alustariippumattomilla kehitystyökaluilla rakennetut sovellukset jaetaan kuitenkin jatkossa yksinkertaistaen natiivisovelluksiin ja tulkattaviin sovelluksiin sen mukaan, käyttävätkö ne ohjelman suorituksen aikana apuna omaa ajoympäristöä. Hybridisovellus on taas erikoistapaus tulkattavasta sovelluksesta, jossa ajoympäristönä toimii natiivisovelluksen WebView-näkymä.

4. ALUSTARIIPPUMATTOMIEN KEHITYSTYÖKALUJEN VALINTA- JA ARVIOINTIKRITEERIT

4.1. Minimivaatimukset alustariippumattomalle kehitystyökalulle

4.1.1. Kohdealustat

Ennen testattavien alustariippumattomien kehitystyökalujen valintaa oli määritettävä niiltä edellytettävät minimivaatimukset. Tätä varten tehtiin lista mobiilikäyttöjärjestelmistä ja niiden versioista, joita työkalulla luodun sovelluksen tulisi tukea (taulukko 4.1).

Taulukko 4.1: *Kehitystyökalulta vaadittava tuki mobiilikäyttöjärjestelmille (* = ei välttämätön)*

Käyttöjärjestelmä	Alin tuettu versio
Android	2.3 (3.x *)
iOS	5.0
Windows Phone *	-

Päätös tuettavien käyttöjärjestelmien ja käyttöjärjestelmäversioiden suhteen tehtiin kohdassa 2.2 läpikäydyn markkinatilannekatsauksen ja kohdassa 2.3 esitettyjen Notes Traveler -sovelluksen käyttäjätilastojen perusteella. Näistä Notes Traveler -tilastot vaikuttivat suuremmalla painoarvolla päätöksentekoon, koska niissä tarkasteltavana olivat Metson MAC-segmentin mobiililaitteet.

Android ja iOS ovat selkeästi suosituimmat mobiilikäyttöjärjestelmät niin kuluttajamarkkinoilla kuin Metson MAC-segmentin käyttäjien kesken, minkä vuoksi työkalun tuki molemmille alustoille on erittäin tärkeää. Vaikka Symbian on edelleen suosittu käyttöjärjestelmä Metso-käyttäjien mobiililaitteissa, voidaan sen suosion odottaa laskevan merkittävästi laitekannan uusiutuessa. Tämän vuoksi päätettiin, että tuki sille ei olisi tulevaisuutta ajatellen tarpeellinen. Windows Phonen suosio on tällä hetkellä hyvin marginaalinen sekä kuluttajamarkkinoilla että Metso-käyttäjien kesken, mutta ennusteiden mukaan sille on odotettavissa huomattavaa kasvua. Tämän vuoksi päätettiin, että tuki Windows Phonelle olisi toivottavaa, mutta ei vielä tällä hetkellä välttämätöntä.

Tukea web-alustalle pidettiin erinomaisena lisänä, koska se laajentaa tuettujen laitteiden määrää kaikkiin niihin laitteisiin, joissa on asennettuna yhteensopiva selain. Tällöin niiden käyttö ei ole myöskään rajattu pelkästään mobiililaitteisiin. Web-sovellukset eivät

kuitenkaan tue laiteominaisuuksia samalla tavoin kuin laitteelle asennettavat sovellukset, minkä vuoksi ne eivät sovi kaikkiin käyttötarkoituksiin. Täten tukea web-alustalle ei pidetty välttämättömänä.

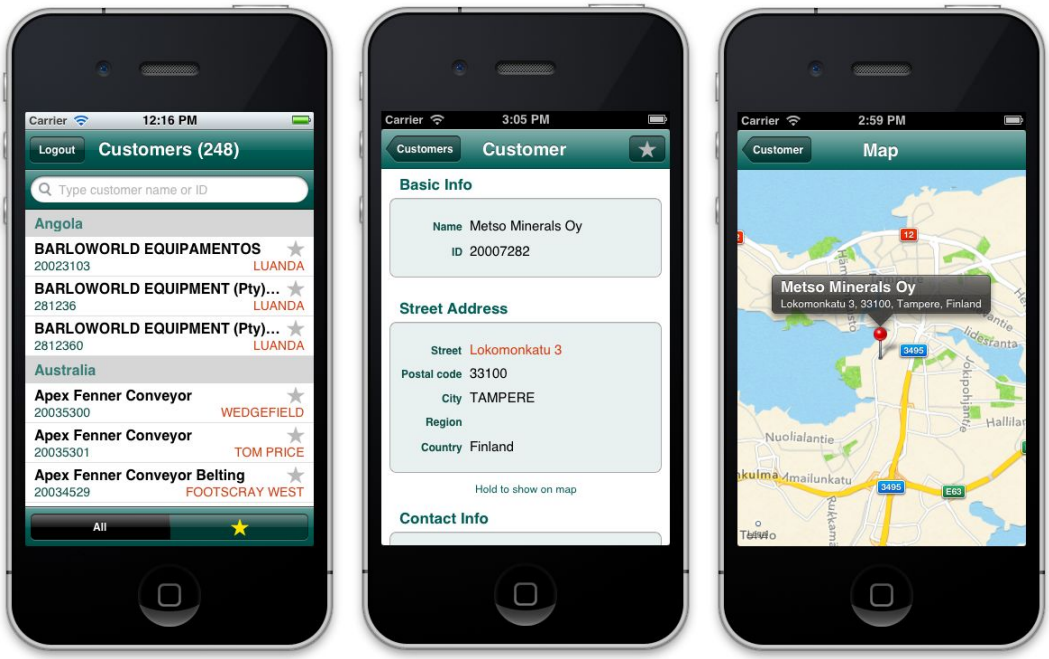
Tuettavat käyttöjärjestelmäversiot valittiin Notes Traveler -käyttäjätilastojen perusteella siten, että kyseiset versiot kattaisivat vähintään 95 prosentin osuuden kyseisen alustan käyttäjistä. Täten alimmaksi tuetuksi Android-versioksi valittiin 2.3 ja iOS-versioksi 5.0. Tukea Android 3.x-käyttöjärjestelmäversiolle ei kuitenkaan pidetty välttämättömänä, koska sen osuus Android-laitteista oli marginaalinen. Windows Phone -käyttäjiä oli Notes Traveler -tilastoissa niin vähän, että tilastoja sen käyttöjärjestelmäversioiden jakaumasta ei pidetty tarpeellisena selvittää. Jos työkalu ylipäätään tukee Windows Phonea, voidaan sitä pitää myönteisenä lisänä.

Käyttöjärjestelmävaatimusten jälkeen rajattiin, mitä laiteominaisuuksia alustariippumattomalla kehitystyökalulla luodun sovelluksen tulisi tukea. Tärkeänä pidettiin sitä, että työkalulla luotu sovellus tukee mahdollisimman joustavasti kaikkia mobiililaitteiden erilaisia näyttökokoja. Muita laiteominaisuuksia koskevia minimivaatimuksia varten laadittiin toiminnallisen vaatimusmäärittelyn esimerkkisovellukselle.

4.1.2. Laiteominaisuudet ja muut toiminnallisuudet

Esimerkkisovelluksen avulla haluttiin testata mahdollisimman useaa todennäköisesti myös tulevissa sovelluksissa tarvittavaa laiteominaisuutta. Nykyiset tai tiedossa olleet tulevat mobiiliprojektit eivät vaikuttaneet kuitenkaan tähän sopivilta, joten esimerkkisovelluksen toiminnallisen vaatimusmäärittely päätettiin tehdä jo olemassa olevan mobiilisovelluksen pohjalta. Lisäksi haluttiin tutkia mahdollisuuksia kyseessä olevan sovelluksen viemisestä iOS-alustalta muille mobiilialustoille.

Alkuperäisen Customers-nimisen iOS-sovelluksen avulla käyttäjät voivat hakea Metson MAC-segmentin asiakas- ja kontaktitietoja SAP-toiminnanohjausjärjestelmästä (kuva 4.1). Sovellusta on mahdollista käyttää yhteydettömässä tilassa, koska yhteydellisessä tilassa haetut tiedot tallennetaan laitteen lokaaliin tietokantaan. Tästä ominaisuudesta on hyötyä erityisesti paljon matkustaville myyntimiehille, jotka eivät ole aina verkon kantaman sisällä. Sovelluksessa hyödynnetään myös laitteen osoitekirjaa ja karttasovellusta.



Kuva 4.1: *Alkuperäinen Customers iOS-sovellus*

Esimerkkisovelluksena toteutetaan niin sanottu rautalankaversio Customers-alkuperäissovelluksesta Android- ja iOS-kohdealustoille. Sovellusta ei ole tarkoitus julkaista testauksen jälkeen, mutta sitä voidaan käyttää lähtökohtana jatkokehitystä ajatellen. Vaadittavat toiminnallisuudet on rajattu siten, että niiden avulla voidaan testata riittävästi haluttuja laiteominaisuuksia toteuttamatta alkuperäisversion toiminnallisuuksia täydessä laajuudessaan. Taulukkoon 4.2 on kuvattu esimerkkisovellukselta vaadittavat toiminnot sekä niiden avulla testattavat laiteominaisuudet ja toiminnallisuudet.

Taulukko 4.2: *Esimerkkisovellukselta vaadittavat toiminnot sekä niiden avulla testattavat laiteominaisuudet ja toiminnallisuudet*

Toiminnon kuvaus	**Testattava laiteominaisuus / toiminnallisuus**
Asiakastiedon (nimi, id, osoite, puhelinnumero, email) haku palvelimelta asiakkaan nimen avulla	Tiedonsiirto verkkoyhteyden yli
Haetun tiedon tallennus laitteelle (suosikkeihin) yhteydetöntä käyttöä varten	Lokaali tietokanta
Asiakkaan kontaktietojen tallennus laitteen osoitekirjaan	Osoitekirja
Asiakkaan ja käyttäjän oman sijainnin esittäminen kartalla	GPS ja karttasovellus

Lokaalia tietokantaa pidettiin heti verkkoyhteyksien jälkeen tärkeimpänä testattavana ominaisuutena, koska sille tulee olemaan todennäköisimmin käyttökohteita myös tulevissa mobiilisovellusprojekteissamme. Sijainnin esittäminen kartalla ja kontaktitietojen

lisäys laitteen osoitekirjaan ovat enemmän esimerkkisovellukselle tunnusomaisia toimintoja, vaikka niillä saattaa olla käyttökohteita myös myöhemmin.

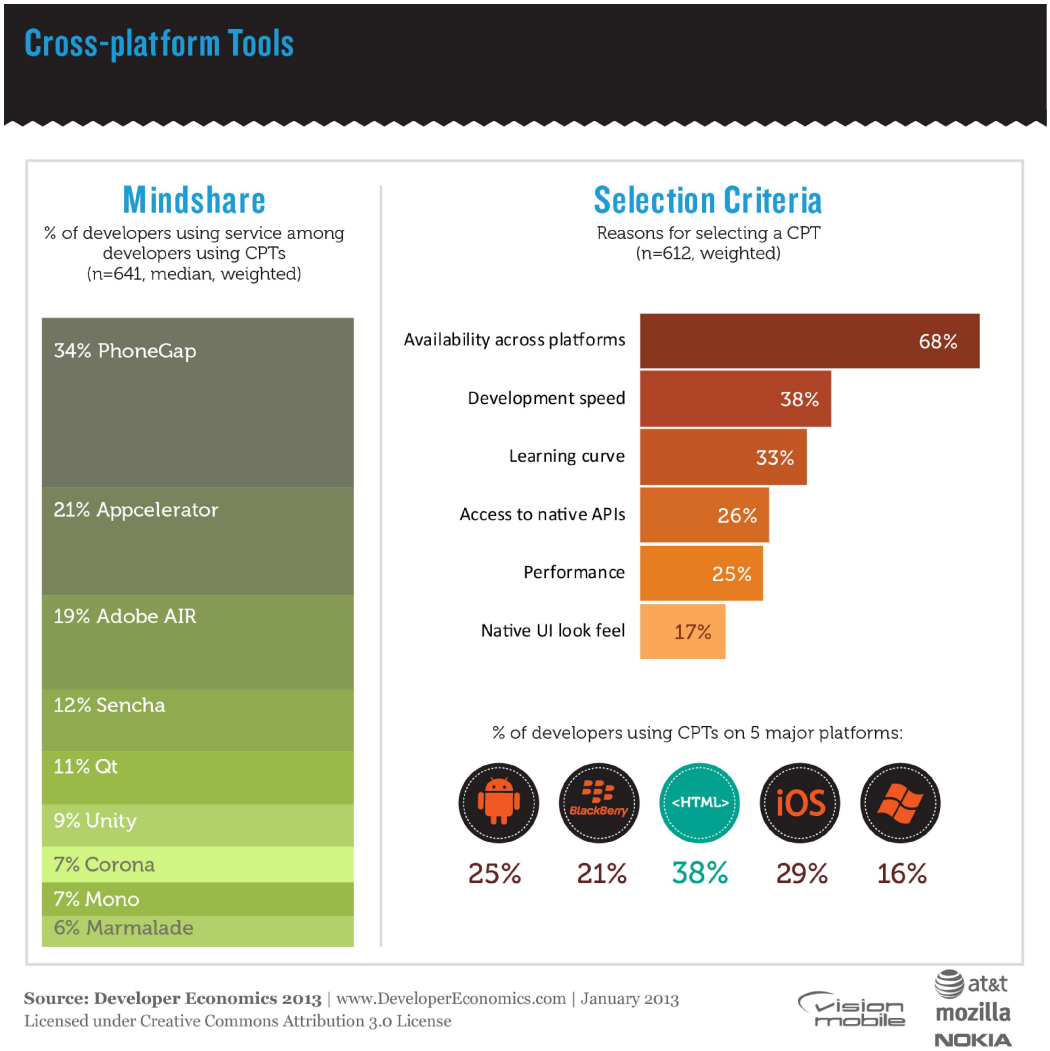
4.2. Kehitystyökalujen vertailu

Olemassa oleviin alustariippumattomiin työkaluihin ja toteutustekniikoihin tutustuttiin kirjallisuuskatsauksen avulla. Tämän tutkimustyön löydöksistä kirjoitettiin yhteenveto lukuun 3. Tavoitteena oli valita viisi työkalua niiden suosioon, arvioituun soveltuvuuteen ja mielenkiintoisuuteen perustuen. Näitä viittä työkalua vertailtiin kohdealustatuen ja esimerkkisovelluksen asettamien laiteominaisuusvaatimusten suhteen. Vertailun perusteella valittiin lopulta kolme tässä työssä testattavaa työkalua.

Vision Mobilen vuoden 2013 tammikuussa julkaisema maailmanlaajuiseen kyselyyn pohjautuva raportti (VisionMobile: Developer Economics 2013) antoi hyvin osviittaa kehitystyökalujen välisestä keskinäisestä suosiosta. Kyselyyn oli vastannut vuoden 2012 lopussa yli 3400 ohjelmistokehittäjää. Kuvassa 4.2 on esitetty Vision Mobilen kyselyn perusteella yhdeksän suosituinta alustariippumatonta kehitystyökalua ja alustariippumattomien työkalujen käyttöönottoon vaikuttaneet tärkeimmät valintakriteerit.

Vision Mobilen raportin (VisionMobile: Developer Economics 2013) mukaan PhoneGap, Appcelerator ja Adobe AIR ovat kolme selkeästi suosituinta alustariippumatonta kehitystyökalua yli sadan työkalun joukosta. Niiden jälkeen työkalujen kesken on tasaisempaa, koska ero neljänneksi suosituimman Sencha Touchin ja yhdeksänneksi suosituimman Marmaladen välillä on vain kuusi prosenttiyksikköä. Kyselyyn vastanneet ohjelmistokehittäjät pitivät kolmena tärkeimpänä valintakriteerinä alustariippumattoman työkalun valinnassa työkalun tukea usealle kohdealustalle, ohjelmistokehityksen nopeutta ja oppimiskynnystä. (VisionMobile: Developer Economics 2013, s. 44.)

Pelkän suosion lisäksi on otettava huomioon, minkälaisia sovelluksia työkaluilla pääasiassa kehitetään. Esimerkiksi Unityn, Coronan ja Marmaladen fokus on vahvasti pelinkehityksessä, kun taas Qt on tunnettu työpöytä- ja sulautettujen järjestelmien sovelluksista (Unity; Corona; Marmalade; Qt). Tämä ei kuitenkaan automaattisesti tarkoita sitä, että kyseiset kehitystyökalut eivät sovellu yrityskäyttöön suunnattujen mobiilisovellusten toteuttamiseen. Soveltuvuuden voidaan kuitenkin arvioida olevan tähän käyttötarkoitukseen varta vasten räätälöityjä kehitystyökaluja heikompi.



Kuva 4.2: *Suosituimmat alustariippumattomat työkalut, alustariippumattomien työkalujen käyttöönottoon vaikuttaneet tärkeimmät valintakriteerit ja kohdealustojen jakauma (VisionMobile: Developer Economics 2013)*

Työkalujen lisenssien hintoja ei pidetty tässä vaiheessa tärkeänä valintakriteerinä. Vaatimuksena kuitenkin oli, että työkalun kokoversiota pääsisi testaamaan riittävän pitkän ilmaisen kokeilujakson ajan. Tämä ajanjakso määritettiin 30 päiväksi, joka on hyvin yleinen kokeilujakson kesto tietokonesovelluksissa.

Viisi mielenkiintoisinta kehitystyökalua valittiin vertailtavaksi niiden suosion ja arvioidun soveltuvuuden perusteella:

- Adobe AIR
- Appcelerator Titanium
- PhoneGap
- Sencha Touch
- Xamarin (Android/iOS).

Näistä Adobe AIR, Appcelerator Titanium ja Xamarin edustavat tulkattavia sovelluksia tuottavia työkaluja, kun taas PhoneGap ja Sencha Touch -työkalujen avulla selaimella

ajettava web-sovellus voidaan paketoida natiivisovelluksen kuoren sisään hybridisovellukseksi.

PhoneGap ei muista valituista työkaluista poiketen tarjoa kokonaisvaltaista mobiilisovellusten kehitysympäristöä, vaan kehittäjän on itse valittava, mitä käyttöliittymä- ja sovelluslogiikkakirjastoa hän haluaa käyttää. Tämä voi olla joko hyvä tai huono asia antaen kehittäjälle enemmän valinnanvaraa eri komponenttien suhteen tai monimutkaistaen toteutusta. PhoneGapin kanssa voidaan käyttää yhdessä esimerkiksi jQuery Mobile -käyttöliittymäkirjastoa, jolla on laaja tuki eri alustojen selaimille. Myös Sencha Touchin oman web-natiivi-käärijän voidaan tarvittaessa korvata PhoneGapilla. (McWherter & Gowell 2012, s. 327–328.) Tällöin Sencha Touch -kirjaston avulla toteutettuja sovelluksia voidaan paketoida myös muun muassa Windows Phone ja BlackBerry -alustoille. Valittuja työkaluja vertailtiin minimivaatimuksissa rajatun kohdealustatuen (taulukko 4.3) ja esimerkkisovelluksen vaatimien laiteominaisuuksien (taulukko 4.4) suhteen.

Taulukko 4.3: *Viiden valitun alustariippumattoman kehitystyökalun tuki minimivaatimuksissa rajatuille kohdealustoille (* = PhoneGapin avulla) (Adobe AIR Dev Center; Appcelerator Titanium Docs; PhoneGap Features; Sencha Touch Docs; Xamarin Dev Centre)*

	Adobe AIR (3.7)	Appcelerator Titanium (3.1.0)	PhoneGap (2.9.0)	Sencha Touch (2.2.1)	Xamarin (2.0)
Android (2.3)	X	X	X	X	X
iOS (5.0)	X	X	X	X	X
Windows Phone			X	X*	X
Web		X	X	X	

Kaikki vertailun kehitystyökalut tukivat minimivaatimuksissa rajattuja Android- ja iOS-kohdealustoja. Tuki Windows Phone -alustalle löytyi PhoneGapista ja Xamarinista. Myös Sencha Touchilla voidaan rakentaa sovelluksia Windows Phone -alustalle, kunhan sovellus kääritään PhoneGapin avulla. Sencha Touchin lisäksi myös Appcelerator Titaniumin avulla voidaan toteuttaa selaimella toimivia web-sovelluksia.

Vertailluista työkaluista heikoin alustatuki oli Adobe AIR:illa, joka tuki tällä hetkellä vain Android- ja iOS-alustoja. Vuoden 2011 lopulla Adobe päätti luopua kokonaan Flash-selainlaajennuksen kehityksestä mobiilialustoille, jonka vuoksi Adobe AIR:illa tuotetut Flash-sovellukset toimivat enää ainoastaan työpöytäkoneiden selaimilla (Winokur 2011). Tämän vuoksi sen ei katsottu tukevan web-alustaa täysin.

Taulukko 4.4: *Viiden valitun alustariippumattoman kehitystyökalun tuki esimerkkisovelluksen vaatimille laiteominaisuuksille (* = PhoneGapin avulla) (Adobe AIR Dev Center; Appcelerator Titanium Docs; PhoneGap Features; Sencha Touch Docs; Xamarin Dev Centre)*

	Adobe AIR (3.7)	Appcelerator Titanium (3.1.0)	PhoneGap (2.9.0)	Sencha Touch (2.2.1)	Xamarin (2.0)
Verkko	X	X	X	X	X
Lokaali tietokanta	X	X	X	X	X
Osoitekirja		X	X	X*	X
GPS ja kartta	X	X	X	X	X

Esimerkkisovelluksen määräämien laiteominaisuuksien ja muiden toiminnallisuuksien tuen kannalta työkaluissa oli vain vähäisiä eroja. Adobe AIR ei tukenut osoitekirjan lukua tai siihen kirjoittamista ilman erikseen ladattavaa epävirallista laajennusta. Sencha Touch taas tuki osoitekirjan lukua, mutta ei siihen kirjoittamista, mikä oli vaadittu toiminnallisuus esimerkkisovelluksessa. Osoitekirjaan kirjoittaminen on kuitenkin mahdollista myös Sencha Touch -sovelluksilla, jos sovellus kääritään PhoneGapin avulla käyttäen hyväksi sen tarjoamia rajapintoja.

4.3. Kehitystyökalujen valinta

Vertailun tulosten perusteella Adobe AIR:n päädyttiin rajaamaan pois varsinaiseen testiin kaavailtujen työkalujen joukosta sen muita työkaluja heikomman alusta- ja laiteominaisuustuen vuoksi. Muiden työkalujen osalta valinta oli huomattavasti vaikeampi. Jäljellä olevista neljästä vaihtoehdosta haluttiin testattavaksi vähintään yksi hybridisovelluksia ja vähintään yksi tulkattavia sovelluksia tuottava työkalu.

Hybridisovelluksia tuottavista työkaluista PhoneGap, jolla on erittäin kattava kohdealusta- ja laiteominaisuustuki, ei sisällä omaa käyttöliittymä- tai ohjelmalogiikkakirjastoa mobiilisovellusten toteutusta varten. Tämän vuoksi sen käyttäminen yksinään ei ole varteenotettava vaihtoehto. Sen sijaan Sencha Touch, joka tarjoaa PhoneGapia kokonaisvaltaisemman kehityspaketin, ei tue yhtä useaa kohdealustaa tai laiteominaisuutta kuin PhoneGap. Sencha Touchin oman web-natiivi-käärijän voi kuitenkin korvata PhoneGapilla, jolloin työkaluilla toteutetun sovelluksen alusta- ja laiteominaisuustuki laajenee. Sencha Touch valittiinkin PhoneGapin kumppaniksi varsinaiseen testiin.

Tulkattavia sovelluksia tuottavien Appcelerator Titaniumin ja Xamarinin välillä ratkaisu tehtiin alustatuen ja ohjelmointikielen perusteella. Vaadittavan Android ja iOS -tuen lisäksi Titanium tukee web-alustaa ja Xamarin Windows Phonea, joista web-alustatukea pidettiin tällä hetkellä Windows Phone -tukea hyödyllisempänä. Vaikka esimerkkisovel-

luksen kannalta kaikkia laiteominaisuuksia ei voisi web-sovelluksessa toteuttaakaan, on potentiaalisten kohdelaitteiden määrä web-alustalla kuitenkin huomattavasti suurempi kuin Windows Phone -kohdelaitteiden määrä. Tämän lisäksi Titaniumin JavaScript-ohjelmointikieli oli ennestään tuttu toisin kuin Xamarinin C#-ohjelmointikieli, minkä vuoksi lopullinen valinta oli Titanium. Titaniumin Windows Phone -tuki on määrä julkaista kuluvan vuoden aikana (Feloney 2013). Valitut alustariippumattomat kehitystyökalut on esitetty taulukossa 4.5.

Taulukko 4.5: *Testattavaksi valitut alustariippumattomat kehitystyökalut*

Kehitystyökalu	Versionumero
Appcelerator Titanium	3.1.1
PhoneGap	2.9.0
Sencha Touch	2.2.1

Esimerkkisovellus toteutetaan Android- ja iOS-alustoille kunkin valitun alustariippumattoman kehitystyökalun avulla. Sencha Touchia käytetään käyttöliittymä- ja sovelluslogiikkakirjastona oman toteutusprojektin lisäksi myös PhoneGap-projektissa. Jokaisesta kehitystyökaluista käytetään testin aikana tuoreinta julkaistua versiota.

4.4. Kehitystyökalujen arviointikriteerit

Toteutusprosessia ja sen lopputuoteita arvioidaan taulukoissa 4.6 ja 4.7 esitettyjen kriteerien avulla. Tarkempi kuvaus siitä, kuinka mittaukset ja arvioinnit suoritettiin, on esitetty luvussa 6.

Taulukko 4.6: *Mittaamalla arvioitavat kriteerit*

Arvioitava kriteeri	Paremmuusjärjestys
Sovelluksen asennuspaketin koko	Pienempi parempi
Sovelluksen keskusmuistin käyttö	Pienempi parempi
Koodirivien lukumäärä	Pienempi parempi
Koodin uudelleenkäytettävyys	Suurempi parempi
Lisenssi (kustannukset)	Pienempi parempi

Mittaamalla arvioitavissa kriteereissä kehitystyökalut laitetaan paremmuusjärjestykseen sen mukaan, miten ne pärjäävät kunkin kriteerin kohdalla toisiinsa verrattuna. Paras ratkaisu ansaitsee kolme pistettä, keskiverto kaksi ja heikoin yhden pisteen. Jos paremmuudessa ei päästä yksimielisyyteen, esimerkiksi kahdella kehitystyökalulla on ilmainen lisenssi ja yhdellä maksullinen, myöntämättä jääneistä sijoituksista saatavat pisteet jaetaan tasan tasavertaisten työkalujen kesken.

Taulukko 4.7: *Subjektiivisesti arvioitavat kriteerit*

Arvioitava kriteeri	**Selite**
Työkalun opittavuus	Minkälainen oppimiskäyrä työkalulla on?
Työkalun dokumentaation kattavuus	Kuinka kattava ja ajan tasalla työkalun dokumentaatio on?
Esimerkkisovelluksen onnistuminen	Saatiinko kaikki esimerkkisovellukselta vaaditut toiminnallisuudet toteutettua työkalun avulla? Jäikö lopputuloksesta jotain toivomisen varaa?
Esimerkkisovelluksen käyttöliittymän sulavuus	Kuinka nopeasti ja tarkasti käyttöliittymä ottaa vastaan käyttäjän syötteitä? Kuinka sulavia animaatiot ovat?

Subjektiivisesti arvioitavien kriteerien kohdalla kehitystyökalut asetetaan paremmuusjärjestykseen käyttäen samaa pisteytystä kuin mittaamalla arvioitavien kriteerien kohdalla. Työkalujen arviointi tehdään nojaten omiin aikaisempiin kokemuksiin vastaavanlaisista työkaluista.

5. ALUSTARIIPPUMATTOMIEN KEHITYSTYÖKALUJEN TESTAUS

5.1. Testialusta, testilaitteet ja web-palvelu

Testialustana käytettiin Mac OS X 10.7.5 -käyttöjärjestelmällä toimivaa MacBook Pro -kannettavaa. Kannettavan 2,7 gigahertsinen Intel Core i7 -suoritin ja 8 gigatavua keskusmuistia ylittivät reilusti kehitystyökalujen asettamat laitteistovaatimukset. Kiintolevytilaa oli koko testijakson ajan vapaana yli 100 gigatavun verran.

Ennen testattavien kehitystyökalujen asennusta huolehdittiin siitä, että kohdealustojen sovelluskehityspaketit olivat asennettuina, koska tämä oli jokaisen työkalun esivaatimuksena. Asennetut Android SDK:n komponentit ja niiden versionumerot on esitetty taulukossa 5.1. Vastaavat iOS SDK:n komponentit ja niiden versionumerot on esitetty taulukossa 5.2.

Taulukko 5.1: *Asennettujen Android SDK -komponenttien versionumerot*

Android SDK -komponentti	Versionumero
SDK Tools	22.0.1
SDK Platform-tools	17
SDK Build-tools	17
SDK Platform (API 17)	4.2.2 revisio 2
Google APIs (API 17)	4.2.2 revisio 3

Taulukko 5.2: *Asennettujen iOS SDK -komponenttien versionumerot*

iOS SDK -komponentti	Versionumero
iOS SDK	6.1
Xcode	4.6.2
Xcode Command Line Tools	4.6.2

Testilaitteina käytettiin Android 4.1.2 -käyttöjärjestelmällä toimivaa Samsung Galaxy S 3 -älypuhelinta ja iOS 6.1.3 -käyttöjärjestelmällä toimivaa kolmannen sukupolven iPad -tablettia. Toimivuutta alakohdassa 4.1.1 määritellyille alimmille tuetuille versioille ei testattu.

Aiempien mobiilisovellusten käytössä ollut rajapinta SAP-kyselyille korvattiin helppokäyttöisemmällä REST-palvelulla, joka haluttiin ottaa käyttöön myös esimerkkisovel-

lusta laajemmin. Esimerkkisovellus käytti REST-palvelua asiakkaan nimeen pohjautuvaa asiakashakua varten.

Seuraavissa kohdissa on kuvattu esimerkkisovelluksen toteutusprosessia valituilla alustariippumattomilla kehitystyökaluilla.

5.2. Sencha Touch

5.2.1. Kuvaus

Sencha Touch on MVC-arkkitehtuurityyliä noudattava HTML5 JavaScript-kehys mobiilien web-sovellusten kehitykseen. Sen kehityksestä vastaa sama työryhmä, joka on aiemmin luonut työpöytäselaimille suunnatun Ext JS -nimisen JavaScript-kehyksen. Ensimmäinen 1.0-versio Sencha Touchista julkaistiin vuoden 2010 lopulla ja Sencha Touch 2 maaliskuussa 2012. (Kosmaczewski 2013, s. 1, 3.)

Sencha Touch sisältää mobiilien web-sovellusten kehityksen kannalta kaiken tarpeellisen, kuten laajan käyttöliittymäkirjaston, sisäänrakennetut liitännät verkon datapalveluille sekä tuen verkkoyhteydettömälle käytölle. Web-sovellusten kehittäminen sen avulla ei vaadi siten muiden kehysten käyttöönottoa. Mobiilit web-sovellukset saa lisäksi pakattua hybridisovelluksiksi Android- ja iOS-alustoille Sencha Cmd -nimisen web-natiivi-käärijän avulla. (Kosmaczewski 2013, s. 3.)

Aiemmin Sencha Touch tuki vain WebKit-selainmoottorilla varustettuja selaimia kuten Google Chromea ja Safaria, minkä vuoksi se pystyi hyödyntämään eksklusiivisesti kyseiselle modernille selainmoottorille ominaisia kehittyneitä web-ominaisuuksia. Sen avulla kehitetyt web-sovellukset eivät tämän vuoksi toimineet muita selainmoottoreita käyttävillä selaimilla kuten Internet Explorerilla tai Mozilla Firefoxilla, mikä rajoitti sen mobiililaitetuen Android- ja iOS- sekä WebKit-selaimen sisältäviin BlackBerry-laitteisiin. (Kosmaczewski 2013, s. 2.) Version 2.2 myötä WebKit-eksklusiivisista ominaisuuksista luovuttiin ja Sencha Touch tukee nyt myös Internet Explorer 10 -selainta, joka on käytössä muun muassa Windows 8 ja Windows Phone 8 -alustoilla. (Sencha.)

Sencha Touch on saatavilla yksittäin sekä osana Sencha Complete ja Sencha Touch Bundle -paketteja. Sencha Touch Bundle sisältää Sencha Touch -kehyksen lisäksi Sencha Architect -nimisen ohjelmointiympäristön, Eclipse-laajennuksen, Sencha Cmd -web-natiivi-käärijän, Sencha Touch Charts -kaaviokirjaston ja käyttäjätukipaketin. Sencha Complete sisältää edellä mainitun lisäksi vielä työpöytäselaimille suunnatun Ext JS -kehyksen ja datayhteyksiä helpottavan kirjaston. (Sencha.) Näistä Sencha Touch Bundle koettiin parhaaksi vaihtoehdoksi esimerkkisovelluksen kannalta ja sitä pääsi kokeilemaan 30 päivän testijakson ajan.

5.2.2. Asennus ja käyttöönotto

Sencha Touch ja Ext JS -kehyksille luotu Sencha Architect -ohjelmointiympäristö on alustariippumaton työpöytäsovellus Windows-, Linux- ja Mac OS X -käyttöjärjestelmille. Se sekä muut Sencha Touch Bundlen -komponentit Eclipse-laajennusta lukuun ottamatta asennettiin testialustalle Senchan sivuilta ladatun asennuspaketin avulla. Asennettujen komponenttien versionumerot on esitetty taulukossa 5.3.

Taulukko 5.3: *Asennettujen Sencha Touch Bundle -komponenttien versionumerot*

Sencha Touch Bundle -komponentti	Versionumero
Sencha Touch	2.2.1
Sencha Architect	2.2.3
Sencha Cmd	3.1.2

Ennen Sencha Architectin käyttöönottoa oli luotava Sencha ID, joka toimii myös käyttäjätunnuksena Senchan yhteisöfoorumeilla. Sencha Touch Bundlen 30 päivän mittainen kokeilujakso oli myös sidottu tähän tunnukseen.

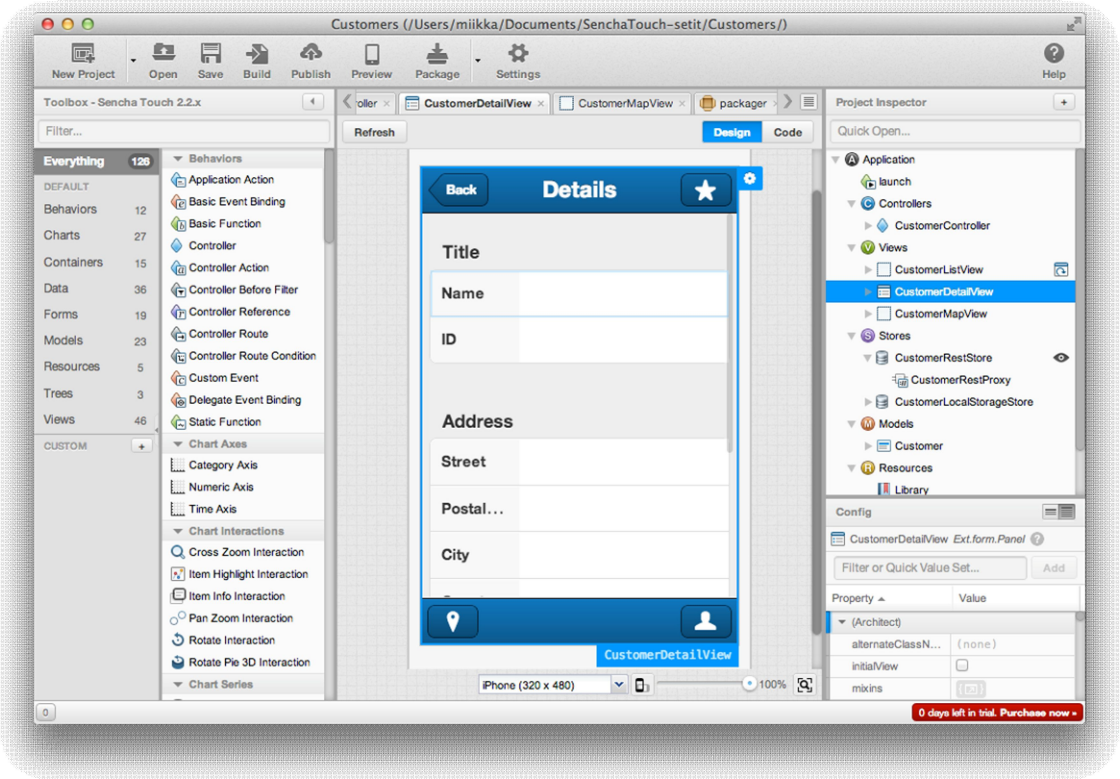
Jos Sencha Architect -ohjelmointiympäristöllä luotua web-sovellusta halusi testata työpöytäselaimelta omalta koneelta, täytyi sitä varten asentaa lokaali web-palvelin. Täten testialustalle asennettiin XAMPP-ohjelmisto. Projektitiedostot siirrettiin lokaalille web-palvelimelle, jonka hakemistopolku ja URL määritettiin Sencha Architectissa.

5.2.3. Sovelluksen toteutus

Sencha Architect oli kevyt ja nopeasti opittava ohjelmointiympäristö. Sen käyttöliittymä oli selkeästi jaoteltu, eikä siihen ollut sullottu liikaa käyttäjiä hämmentäviä toimintoja. Sencha Architect sisälsi lisäksi helppokäyttöisen vedä ja pudota (*engl. drag and drop*) -tekniikalla toimivan työkalun käyttöliittymän visuaaliseen suunnitteluun (kuva 5.1).

Sencha Architect nojasi erittäin paljon generoidun koodin varaan. Uusia malleja, näkymiä, kontrollereja, säilöjä ja resursseja luotiin Inspector-näkymässä, joka esitti hierarkisesti kaikki projektin komponentit. Näitä komponentteja muokattiin Config-näkymässä muunnellen niiden ominaisuuksia tai lisäten niille tapahtumia ja funktioita. Komponenttien ominaisuuksia voitiin tarkastella koodieditorin kautta, mutta muokata sillä pystyi ainoastaan tapahtumakäsittelijöiden ja funktioiden sisältöjä.

Sencha Architectin generoima lähdekoodi oli hyvin siistiä ja organisoitua, mutta esimerkiksi kommentteja ei saanut lisättyä kuin niihin kohtiin, joiden muokkaaminen oli sallittu koodieditorissa. Sencha Architect ei myöskään sisältänyt toimintoa koodin automaattiselle täydennykselle, joka on yleinen käytettävyyttä ja tehokkuutta parantava toiminnallisuus vastaavissa ohjelmointiympäristöissä.



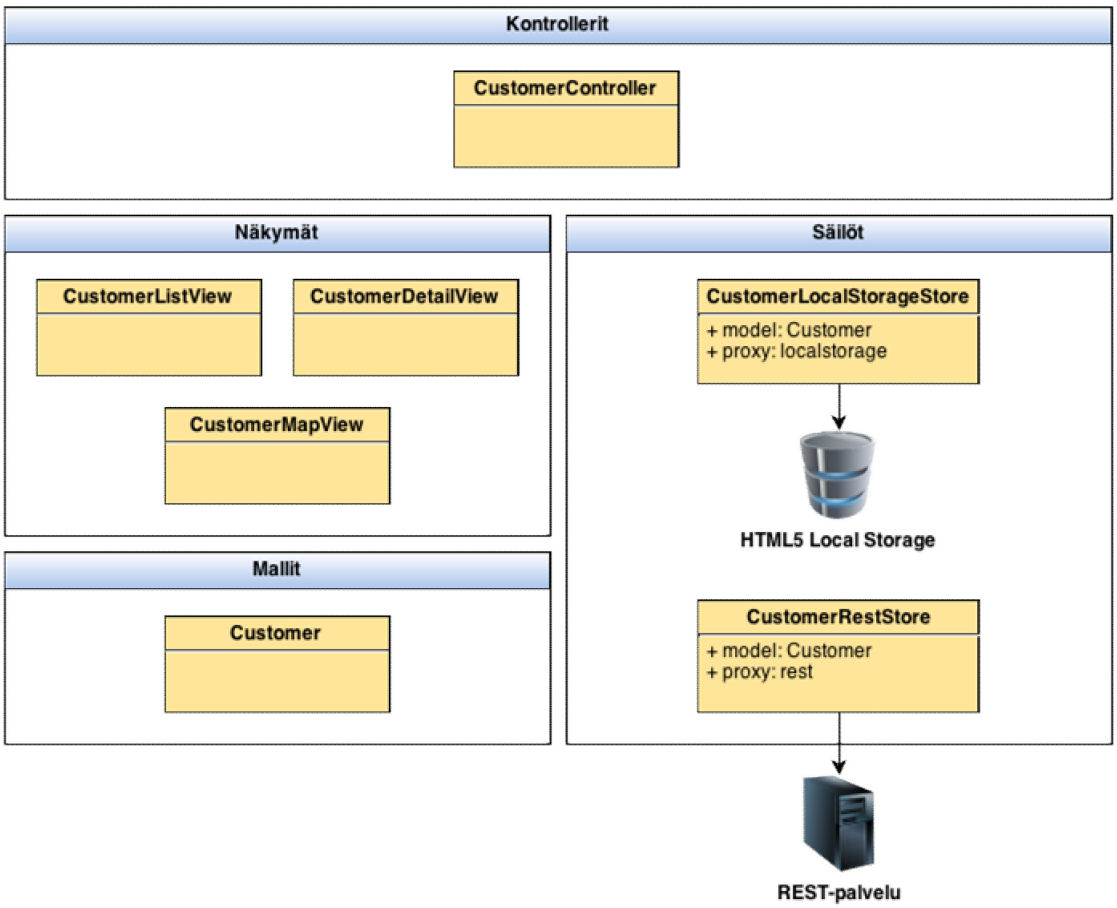
Kuva 5.1: *Sencha Architect*

Sencha Touch -kehys noudattaa sovellusarkkitehtuurissaan omaa muunnelmaa MVC-mallista. Malli-, näkymä- ja kontrolleri-luokkien kanssa samalla hierarkiatasolla ovat säilöt (*engl. store*), jotka sisältävät joukon malli-luokan instansseja. Säilö kytketään tyypillisesti välittäjä-luokkaan (*engl. proxy*), joka kapseloi säilön ja lokaalin tai etätietovaraston välisen datayhteyden. Yleisesti ottaen Sencha Touch -kehyksen MVC-mallin näkymät huolehtivat käyttöliittymän esityksestä, mallit ja säilöt datan käsittelystä ja kontrollerit sovelluslogiikasta. Kuvassa 5.2 on esitetty esimerkkisovelluksen pääluokat jaettuna Sencha Touch -kehyksen MVC-mallin mukaisesti.

Sencha Touch -kehys luo korkean tason abstraktion JavaScript-kielen päälle, minkä ansiosta uusia luokkia voidaan luoda ja olemassa olevia periyttää vastaavalla tavalla kuin muissa olio-ohjelmointikielissä (Kosmaczewski 2013, s. 23). Esimerkiksi uusia käyttöliittymäelementtejä kuten painikkeita voidaan luoda periyttämällä niitä aiemmin luoduista painikkeista. Tämä yhdessä MVC-arkkitehtuurimallin kanssa helpotti uuden kehyksen opettelua, koska yhtäläisyyksiä natiivisovelluskehitykseen oli yllättävän paljon.

Suurinta osaa esimerkkisovelluksen toiminnallisuuksista pystyttiin testaamaan työpöytäselaimen kautta lokaalilla web-palvelimella, mikä oli huomattavasti nopeampaa kuin jos sovellus olisi käännetty jokaisen muutoksen jälkeen ajettavaksi mobiililaitteella. Sencha Architectin Preview-painike latasi viimeisimmin tallennetun version sovelluk-

sesta lokaalille web-palvelimelle, jolloin sovellusta voitiin testata sen URL-osoitteesta WebKit-pohjaisilla Safari tai Google Chrome -työpöytäselaimilla.



Kuva 5.2: *Sencha Touch -kehyksellä rakennetun esimerkkisovelluksen pääluokat jaettuna MVC-mallin mukaisesti*

Simulate-painikkeella esimerkkisovellus voitiin ajaa joko Android-emulaattorilla tai iOS-simulaattorilla kunhan kohdealusta ja -laitetyyppi oli ensin määritelty packager.json-tiedostoon. Pakkausprosessissa Sencha Architect käytti apunaan Sencha Cmd -työkalua.

Kun esimerkkisovellusta ajettiin virtuaaliselta tai oikealta mobiililaitteelta, JavaScriptin virheiden jäljitys (*engl. debugging*) vaikeutui, koska työpöytäselainten kehittäjätyökaluihin ei päässyt enää käsiksi. Tätä varten otettiin käyttöön Weinre-työkalu, jonka avulla mobiilisovellusten selaininstanssin käyttöä voitiin jäljittää etänä (Weinre). Weinren ominaisuudet olivat kuitenkin hieman rajatummat kuin työpöytäselainten vastaavilla kehittäjätyökaluilla.

5.2.4. Sovelluksen jakelu eri mobiilialustoille

Sovelluksen jakelu Android- ja iOS-alustoille Sencha Architectin avulla oli hyvin suoraviivaista. Ensin täytyi määrittää alustakohtaiset asetukset, kuten haluttu Android API-

taso ja iOS-alustan Provision profile -tiedosto, packager.json-tiedostossa. Tämän jälkeen Build-painikkeella käynnistettiin Sencha Cmd -työkalu, joka rakensi asennuspaketin valitulle kohdealustalle sille määritettyyn kansioon.

Esimerkkisovelluksen toteutuksessa ei käytetty lainkaan alustakohtaista ohjelmakoodia. Paremman käyttäjäkokemuksen saavuttamiseksi CSS-tyylitiedostojen luominen eri kohdealustoille on kuitenkin suositeltavaa. Huomioitavaa on, että Sencha Touch ei tukenut laiteominaisuuksista kontaktitietojen lisäystä, minkä vuoksi kyseinen toiminnallisuus jätettiin esimerkkisovelluksesta kokonaan pois.

5.3. PhoneGap

5.3.1. Kuvaus

PhoneGap on avoimen lähdekoodin web-natiivi-käärijä, jonka avulla HTML, CSS ja JavaScript -tekniikoilla toteutettu web-sovellus voidaan kääriä natiivisovelluksen kuorien sisään hybridisovellukseksi. Tällöin sovellus voi käyttää kohdealustan laiteominaisuuksia PhoneGapin tarjoamien JavaScript-rajapintojen kautta. PhoneGap ei ota kantaa siihen, miten sovelluksen käyttöliittymä ja sovelluslogiikka toteutetaan, joten sitä käytetään tyypillisesti yhdessä muiden JavaScript-kehysten kuten jQuery Mobilen tai Sencha Touchin kanssa. PhoneGapin tukemia kohdealustoja ovat Android, iOS, Windows Phone, BlackBerry, webOS, Symbian ja Bada (PhoneGap).

Alun perin Nitobin vuonna 2008 aloittama projekti siirtyi Adoben omistukseen yritysoston myötä vuonna 2011. Nitobin jäsenet, jotka työskentelivät aiemmin projektin parissa ainoastaan vapaa-ajallaan, siirtyivät Adoben palkollisiksi työstämään PhoneGapia kokopäiväisesti. Tämän jälkeen PhoneGap-projekti on saanut tasaiseen tahtiin päivityksiä. (Wargo 2012.)

Kenties merkittävin uudistus PhoneGap-konseptiin on pilvipalvelu PhoneGap Build, jonka avulla käyttäjät voivat lähettää web-sovelluksen HTML-, CSS- ja JavaScript-tiedostot palvelulle, joka rakentaa niistä automaattisesti asennuspaketit halutuille kohdealustoille (PhoneGap Build). PhoneGap Build -pilvipalvelua ei kuitenkaan testattu tämän testijakson aikana, koska sen ilmainen versio ei sallinut muita kuin julkisten, kaikkien saatavilla olevien, projektien lähettämisen palveluun, eikä maksullisestakaan versiosta ollut tarjolla ilmaiskokeilujaksoa.

5.3.2. Asennus ja käyttöönotto

PhoneGapista asennettiin sen hetkinen tuorein julkaistu versio 2.9.0. PhoneGap-projektit luotiin suorittamalla komentoriviltä alla esitetty komento PhoneGapin kohdealustakohtaisessa bin-kansiossa:

```
/create <projektikansion_polku> <pakkauksen_nimi> <projektin_nimi>
```

Android- ja iOS-projektit luotiin eri kansioihin käyttäen samaa pakkauksen ja projektin nimeä. Kun projektit oli luotu, Android-projektin sai avattua ADT-lisäosan sisältävän Eclipse-ohjelmointiympäristön ja iOS-projektin Xcode-ohjelmointiympäristön avulla.

5.3.3. Sovelluksen toteutus

PhoneGap-sovellukset toteutettiin käyttäen ohjelmointiympäristönä Android-projektissa ADT-lisäosan sisältävää Eclipseä ja iOS-projektissa Xcodea, joita käytetään tyypillisesti natiivisovelluskehityksessä. Sovellusten käyttöliittymä ja sovelluslogiikka tuotiin aiemmin rakennetusta Sencha Touch -projektista, koska PhoneGapin tarjoamat rajapinnat liittyvät ainoastaan laiteominaisuuksiin, eikä se sisällä omia käyttöliittymästä tai sovelluslogiikasta vastaavia kirjastoja. Tämä tehtiin kopioimalla Sencha Touch -projektin app-kansio, app.js-, app.json- ja index.html-tiedostot sekä Sencha Touch -kehyksen sisältävä touch-kansio Android-projektissa assets/www- ja iOS-projektissa www-kansion alle. PhoneGap otettiin käyttöön lisäämällä viittaus sen kirjastoon app.json-tiedostossa.

Sencha Touch -projektin vahvoja liitäntöjä REST-palvelun, lokaalin tietokannan ja mallin välillä ei haluttu rikkoa, koska se ei ollut välttämätöntä ja olisi turhaan monimutkaistanut jo toimivaa toteutusta. Sen sijaan kontaktien lisäys osoitekirjaan ja käyttäjän oman sijainnin nouto laitteelta korvattiin käyttäen PhoneGapin JavaScript-rajapintoja. Kaikki toteutetut ominaisuudet olivat tuettuja molemmilla kohdealustoilla, eikä alustakohtaista koodia tarvinnut käyttää lainkaan.

Sovellusta testattiin suoraan testilaitteilla Eclipsen ja Xcoden kautta. JavaScript-koodin virheiden jäljitykseen käytettiin apuna iOS-projektissa Safari-selaimen iOS Web Inspector -työkalua ja Android-projektissa Sencha Touch -projektin tavoin Weinrea. PhoneGap tarjoaa tähän tarkoitukseen myös oman Weinre-palvelimen (PhoneGap Debug).

5.3.4. Sovelluksen jakelu eri mobiilialustoille

PhoneGap-sovellusten jakeluprosessi eri mobiilialustoille ei eronnut natiivisovelluskehityksestä, jossa sovellusten asennuspaketit rakennetaan käyttäen kohdealustojen omia kehitystyökaluja. Tästä on se etu, että kohdealustojen natiivisovelluskehityksen tuntevan kehittäjän ei tarvitse opetella uusien työkalujen käyttöä. Toisaalta jos kehittäjä ei ennestään tunne kohdealustojen kehitystyökaluja, täytyy hänen opetella käyttämään niistä jokaista. Useamman alustakohtaisen projektin ylläpito ja sovellusten jakelu voi täten olla työlästä, vaikka projektit jakaisivatkin saman sovelluskoodin. Olisikin ollut mielenkiintoista nähdä, kuinka hyvin PhoneGap Build olisi ratkaissut tämän ongelman.

5.4. Appcelerator Titanium

5.4.1. Kuvaus

Appcelerator Titanium on JavaScript-kehys mobiilisovellusten kehittämiseen Android-, iOS-, BlackBerry-, Tizen- ja web-alustoille (Appcelerator Titanium Docs). Joulukuussa vuonna 2008 julkaistu Appcelerator Titanium oli alun perin keskittynyt työpöytä- ja web-sovelluksiin, mutta kesäkuussa vuonna 2009 lisätyn Android- ja iPhone-tuen myötä fokus kääntyi mobiilisovelluksiin (McWherter & Gowell 2012, s. 283; Krill 2009). Hybridisovelluksia tuottavista PhoneGap- ja Sencha Touch -kehyksistä poiketen Appcelerator Titaniumissa käyttöliittymä rakennetaan täysin ohjelmallisesti JavaScript-kielen avulla hyödyntäen natiiveja käyttöliittymäelementtejä, eikä HTML5- tai CSS-kieliä käytetä lainkaan. Kun Titaniumilla käännettyä sovellusta ajetaan kohdealustalla, sen JavaScript-moottori huolehtii pyyntöjen tulkkaamisesta kohdealustan ymmärtämään muotoon. (Heitkötter et al. 2012.)

Appcelerator Titanium on saatavilla ilmaisena Titanium-pakettina sekä maksullisena pienille ja suurille yrityksille räätälöitynä Appcelerator Platform -pakettina. Titanium-paketti sisältää kaiken tarpeellisen mobiilisovellusten kehitystä varten mukaan lukien Studio-ohjelmointiympäristön ja Alloy MVC -kehyksen. Ilmaisella versiolla on mahdollista käyttää rajatusti myös maksullisissa paketeissa tarjottuja pilvipalveluita kuten tiedostojakopalvelua. Maksulliset Appcelerator Platform -paketit tarjoavat Titanium-paketin lisäksi kattavammat tuki- ja koulutuspalvelut sekä analytiikka- ja testaustyökalut. (Appcelerator Pricing.) Appcelerator Platform -paketin tarjoamia lisäpalveluita ei kuitenkaan pidetty tarpeellisina esimerkkisovelluksen kannalta, joten testattavaksi valittiin ilmainen Titanium-paketti.

5.4.2. Asennus ja käyttöönotto

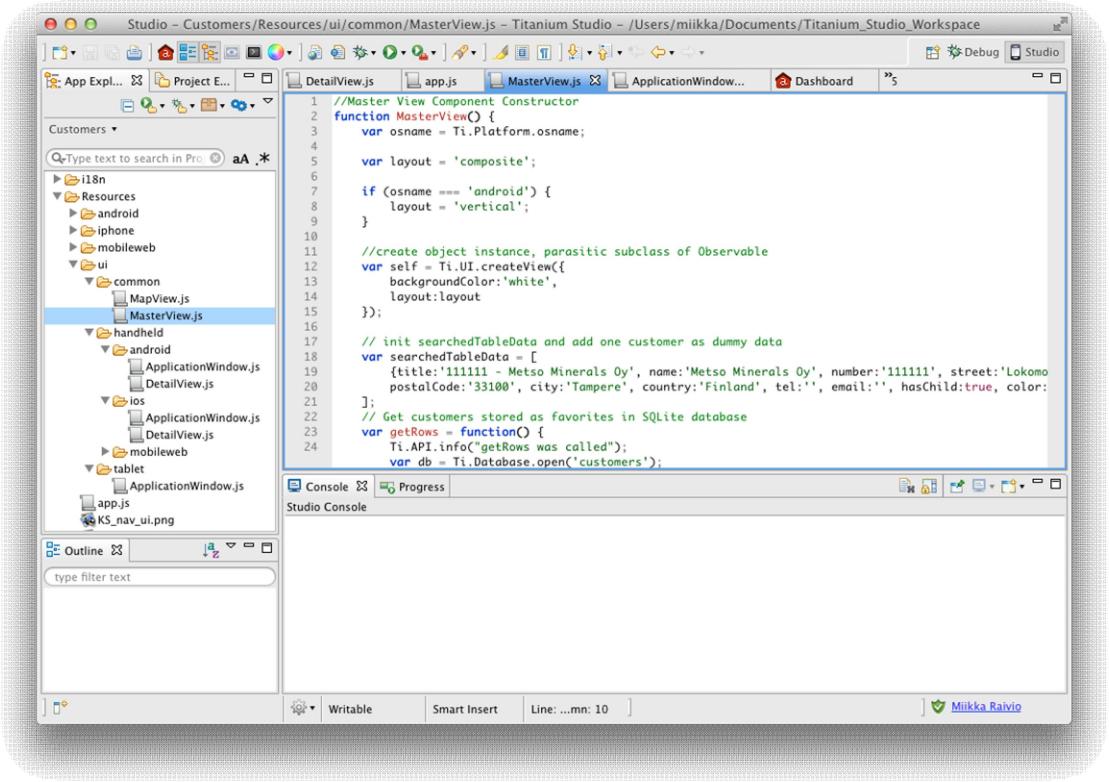
Titanium SDK:sta ja sen ohjelmointiympäristöstä Titanium Studiosta asennettiin versio 3.1.1. Ohjelman ensikäynnistyksellä Studio pyysi kirjautumaan sisään Appcelerator Network -tunnuksilla, jotka sai luotua ilmaiseksi Appceleratorin verkkosivuilla.

Titanium Studio tunnisti asennetun iOS SDK:n automaattisesti, mutta Android SDK:sta oli asennettava testialustalle esiasennetun version lisäksi versio 2.3.3. Tämä oli Titanium SDK:n tukema alin Android-versio. Kun Android SDK oli ensin asennettu, sen hakemisto piti vielä määrittää käsin Studion asetuksiin, minkä jälkeen Studio oli käyttövalmis Android- ja iOS-kehitykseen.

5.4.3. Sovelluksen toteutus

Appcelerator Titaniumin ohjelmointiympäristö Titanium Studio pohjautuu Eclipse-alustaan, minkä huomaa sen yleisilmeestä ja toimintojen kattavuudesta (kuva 5.3). Se sisältää muun muassa toiminnot koodin syntaksin tarkastelulle ja automaattiselle täy-

dennykselle. (Andersson 2013, s. 25.) Ainoa silmiinpistävä heikkous verrattuna vastaaviin ohjelmointiympäristöihin oli visuaalisen käyttöliittymäsuunnittelutyökalun puute, jonka vuoksi käyttöliittymä täytyi rakentaa Studiossa täysin ohjelmallisesti.

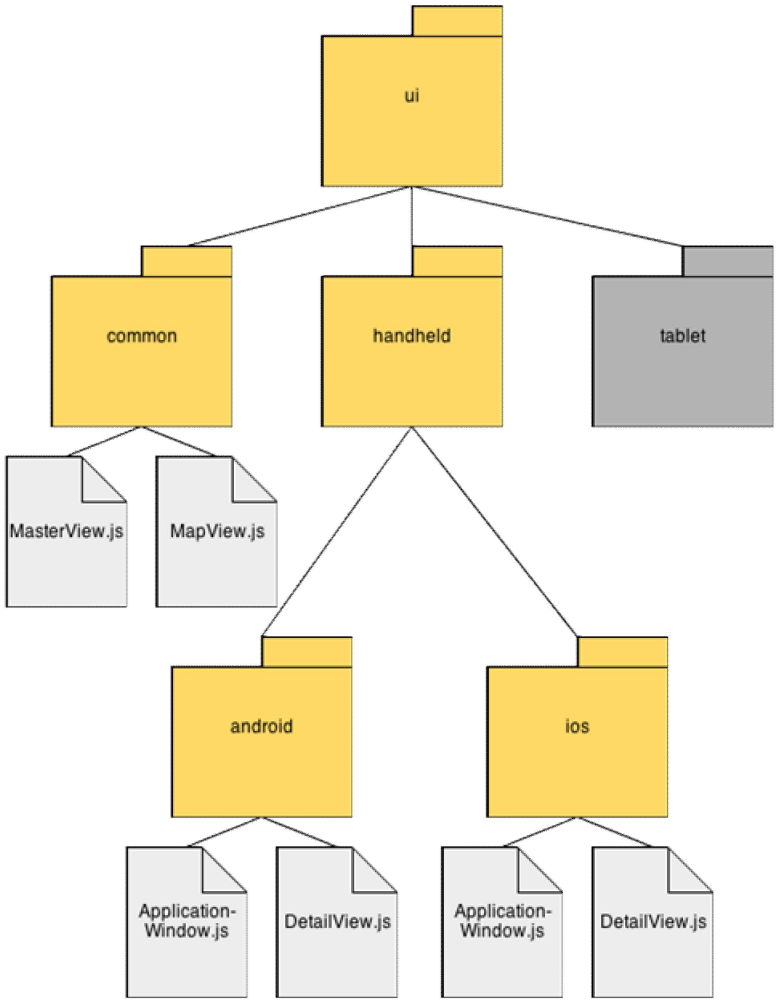


Kuva 5.3: *Titanium Studio*

Titanium tarjosi kaksi vaihtoehtoista tapaa rakentaa uusi mobiiliprojekti: joko käyttäen Alloy MVC -kehystä tai ilman sitä. Näistä vaihtoehdoista valittiin jälkimmäinen, koska tuoreesta Alloy MVC -kehyksestä oli saatavilla siihen nähden huomattavasti vähemmän taustamateriaalia. Alloy MVC -kehyksen etuna olisi ollut ennalta määrätty arkkitehtuurimalli, josta olisi ollut apua koodin organisoinnissa erityisesti aloitteleville JavaScript-ohjelmoijille. Testiprojekti oli kuitenkin sen verran pieni, että Alloy MVC -kehyksen mahdollisesti tuoma etu arvioitiin sen opetteluun kuluvaan aikaan nähden hyvin pieneksi.

Vaikka Appcelerator Titanium käyttää eri alustojen natiivirajapintoja, kehittäjän ei tarvitse tuntea niiden yksityiskohtia, koska Titanium huolehtii viestimisestä niiden kanssa kulissien takana. Suuri osa Titaniumin rajapinnoista on yhteisiä eri alustoilla, esimerkiksi kartta-objekti on Android- ja iOS-alustoilla saman tunnisteen, Ti.Map, takana. Jotkin alustojen käyttöliittymäelementeistä ovat kuitenkin niin erilaisia toiseen alustaan nähden, kuten Android-laitteen fyysisestä Menu-painikkeesta avautuva valikko, että niitä ei ole ollut mahdollista abstrahoida yhteisen rajapinnan taakse. Titanium onkin jakanut alustakohtaiset rajapinnat selkeästi omiin nimiavaruuksiinsa, esimerkiksi Androidin Menu-painikkeen valikon tunniste on Ti.Android.Menu.

Projektin koodi organisoitiin projektipohjan ehdottamalla tavalla (kuva 5.4). Vähän alustakohtaisia ominaisuuksia sisältäneet näkymät toteutettiin common-kansion alle. Näissä näkymissä alustakohtaiset eroavaisuudet voitiin toteuttaa if-else-lauseiden avulla. Näkymät, jotka vaativat huomattavasti alustakohtaisia muutoksia, toteutettiin kokonaan omiin kooditiedostoihinsa android- ja ios-kansioiden alle. Alkuperäinen projektipohja sisälsi erikseen kansion myös tablet-näkymille. Niitä ei kuitenkaan haluttu tässä projektissa toteuttaa erikseen, vaan tablet-laitteet käyttivät samoja näkymiä kuin pieniruutuisemmat älypuhelimet.



Kuva 5.4: *Appcelerator Titanium -projektin sovelluskoodin hakemistorakenne*

Titanium Studion avulla oli mahdollista jäljittää sovellusten virheitä Android-emulaattorin ja iOS-simulaattorin lisäksi oikeilla Android- ja iOS-laitteilla. Studioon oli tätä varten ensin luotava debug-konfiguraatio, johon määritettiin vähintään haluttu kohdealustan SDK-versio. Kun konfiguraatio oli kerran määritetty, se tallennettiin seuraavaa käyttökertaa varten. Titanium Studion avulla JavaScript-koodin virheitä voitiin jäljittää muun muassa asettamalla koodiin pysähdyspisteitä ja tarkkailemalla muuttujien arvoja halutuissa ohjelman suorituskohdissa.

5.4.4. Sovelluksen jakelu eri mobiilialustoille

Asennuspakettien rakentaminen Android- ja iOS-alustoille Titanium Studion avulla oli hyvin suoraviivaista. Studio tarjoaa eri alustojen eri jakelumalleihin omat avustajansa, joita seuraamalla Android- ja iOS-konfiguraatiot saatiin luotua vaivattomasti. Avustaja tallensi kerran asetetut konfiguraatiot, joten ne sai myöhemmin ajettua yhdellä hiiren klikkauksella. Tämän ansiosta esimerkkisovelluksen jakelu samasta projektista eri mobiilialustoille oli erittäin nopeaa.

6. ALUSTARIIPPUMATTOMIEN KEHITYSTYÖKALUJEN ARVIOINTI

6.1. Määrälliset arviointikriteerit

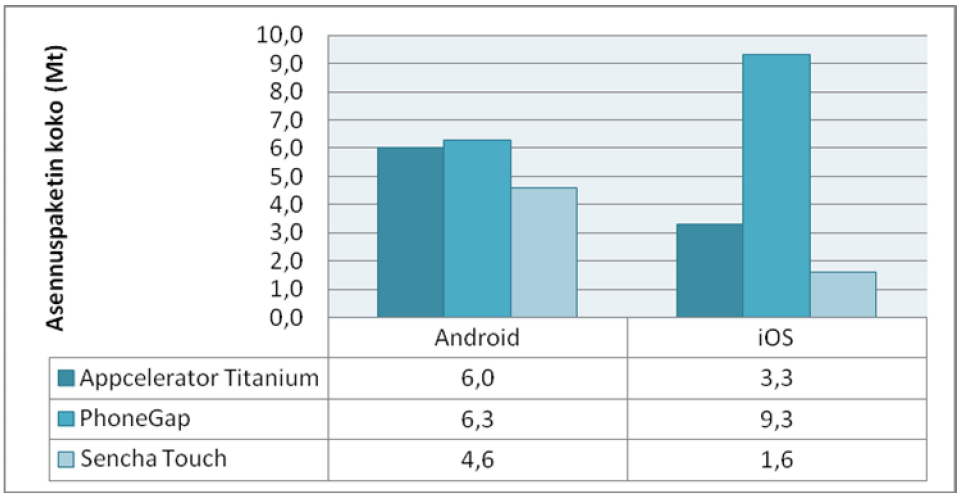
6.1.1. Sovelluksen asennuspaketin koko

Asennuspaketin pienempi tiedostokoko merkitsee lyhyempiä lataus- ja asennusaikoja, joiden tärkeys korostuu etenkin hitaita verkkoyhteyksiä käyttävillä mobiililaitteilla. Pienet erot tiedostokoossa ovat merkityksettömiä, mutta moninkertainen tiedostokoko tarkoittaa moninkertaista lataus- ja asennusaikaa. Applen App Storen vaatimuksena on, että kaikki yli 50 megatavun kokoiset asennuspaketit on ladattava Wi-Fi-yhteyden yli. Tämän vuoksi iOS-sovelluskehittäjät pyrkivät tyypillisesti pitämään sovelluksensa tiedostokoon ainakin alle tämän rajan.

Ennen asennuspakettien rakentamista PhoneGap-projektin sisältöä päätettiin muokata poistamalla siitä kaikki Sencha Touch -kehyksen sisältämät kuvat sekä muita työkaluja tarkemmat iOS-latauskuvat (*engl. splash screen*), koska ne kasvattivat tarpeettomasti PhoneGapin avulla luodun iOS-asennuspaketin kokoa useilla megatavuilla. Tällä haluttiin varmistaa se, että projektien sisältämien kuvatiedostojen koot olisivat mahdollisimman merkityksettömiä vertailtaessa eri työkaluilla rakennettujen asennuspakettien kokoja.

Asennuspaketit rakennettiin testialustalla kullakin kehitystyökalulla Android- ja iOS-kohdealustoille. Testialustan Mac OS X -käyttöjärjestelmän tavuina ilmoittamat tiedostokoot muunnettiin megatavuiksi ja pyöristettiin yhden desimaalin tarkkuudelle. Mitatut asennuspakettien tiedostokoot on esitetty kuvassa 6.1.

Kuvasta 6.1 nähdään, että Sencha Touchin avulla rakennettujen asennuspakettien tiedostokoot olivat molemmilla kohdealustoilla muita työkaluja pienemmät. PhoneGapilla rakennettujen asennuspakettien suurempi koko muihin työkaluihin verrattuna johtui osittain siitä, että PhoneGap-projekti sisälsi oman sovelluskehyksen lisäksi myös Sencha Touch -sovelluskehyksen. Android-asennuspakettien tiedostokokojen erot eivät olleet merkittävän suuria, mutta iOS-alustalla PhoneGap-sovelluksen asennuspaketin tiedostokoko oli lähes kolminkertainen toiseksi suurimpaan Appcelerator Titanium -sovelluksen asennuspakettiin verrattuna.



	Android	iOS
Appcelerator Titanium	6,0	3,3
PhoneGap	6,3	9,3
Sencha Touch	4,6	1,6

Kuva 6.1: *Eri kehitystyökaluilla toteutettujen sovellusten Android- ja iOS-asennuspakettien tiedostokoot*

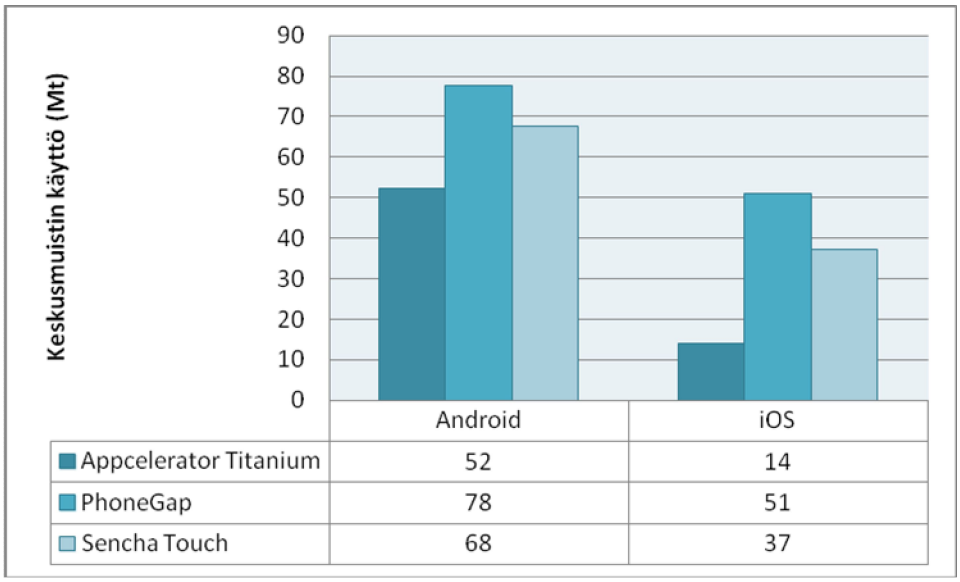
Tulos: Sencha Touch 3 pistettä, Appcelerator Titanium 2 pistettä, PhoneGap 1 piste.

6.1.2. Sovelluksen keskusmuistin käyttö

Nykyisissä mobiilikäyttöjärjestelmissä on mahdollista ajaa useita prosesseja rinnakkain, jolloin prosessit jakavat saman keskusmuistin. Tällöin on tärkeää, että yksi sovellus ei varaa prosessilleen kohtuuttomasti keskusmuistia pakottaen käyttöjärjestelmän sulkemaan muita prosesseja.

Esimerkkisovellusten keskusmuistin käyttöä tarkkailtiin ensimmäisen käynnistyksen jälkeen Android- ja iOS-testilaitteilla. Android-alustalla prosessin keskusmuistin kokonaiskäyttöä (*engl. resident set size, RSS*) mitattiin testilaitteeseen asennetun System Monitor -sovelluksen avulla (System Monitor). iOS-alustalla keskusmuistin käyttöä tarkkailtiin Xcoden Instruments-työkalun Activity Monitor -näkymän Real Memory -sarakkeesta. Mittaustulokset on esitetty kuvassa 6.2.

Appcelerator Titanium -työkalulla luotu sovellus käytti muita vähemmän keskusmuistia molemmilla alustoilla. Ero oli hyvin selvä erityisesti iOS-alustalla, jolla se käytti keskusmuistia vain noin neljänneksen PhoneGap-sovellukseen nähden. PhoneGap-sovelluksen muita suurempaa keskusmuistin käyttöä selittänee sen sisältämät kaksi sovelluskehystä.



	Android	iOS
Appcelerator Titanium	52	14
PhoneGap	78	51
Sencha Touch	68	37

Kuva 6.2: *Eri kehitystyökaluilla toteutettujen sovellusten keskusmuistin käyttö ensimmäisen käynnistyksen jälkeen Android- ja iOS-alustoilla*

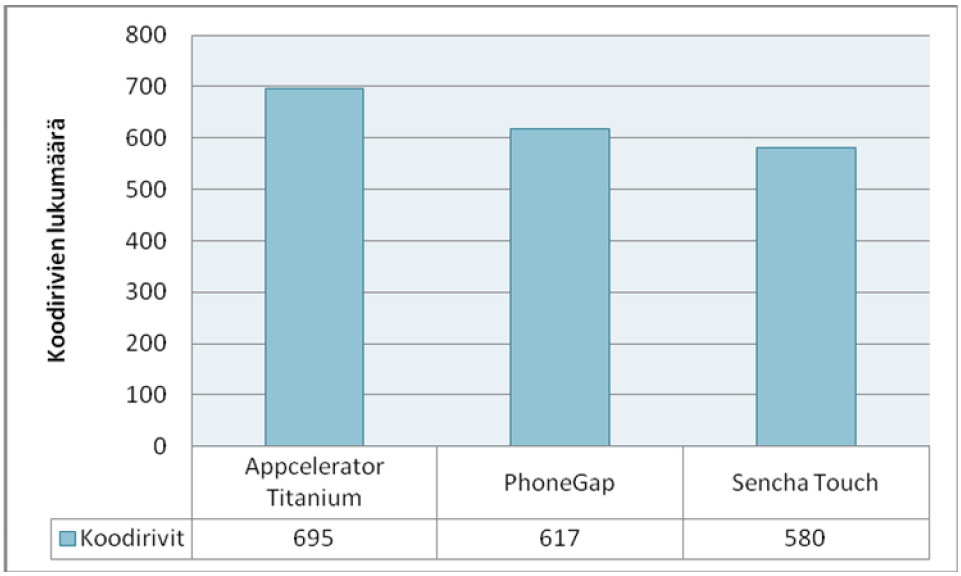
Tulos: Appcelerator Titanium 3 pistettä, Sencha Touch 2 pistettä, PhoneGap 1 piste.

6.1.3. Koodirivien lukumäärä

Vertaamalla kullakin kehitystyökalulla toteutetun esimerkkisovelluksen koodirivien lukumäärää voitiin arvioida kunkin ohjelmistokehyksen ilmaisuvoimaisuutta. Muita pienempi koodirivien lukumäärä viestii, että kyseisellä kehitystyökalulla saa toteutettua saman sovelluksen muita tehokkaammin pienemmällä työmäärällä. Lähdekoodin määrän pienetessä myös sovelluksen ylläpidettävyys helpottuu.

Esimerkkisovellusten koodirivien lukumäärä mitattiin CLOC-nimisen työkalun avulla (CLOC). Mittaustuloksiin otettiin mukaan kaikki tiedostot, jotka sisälsivät itse kirjoitettua lähdekoodia. Asetustiedostot päätettiin jättää kuitenkin mittauksista pois. Kuvassa 6.3 esitettyihin koodirivien määriin ei ole laskettu mukaan tyhjiä eikä kommenttirivejä.

Mittaustulosten perusteella ilmaisuvoimaisin ohjelmistokehys oli Sencha Touch. On kuitenkin otettava huomioon, että sen toteutuksesta puuttui kokonaan toiminnallisuus kontaktitietojen lisäykselle, minkä vuoksi sen mittaustulosta ei voida pitää täysin vertailukelpoisena muiden työkalujen kanssa. PhoneGap-projekti sisälsi Sencha Touch -projektiin nähden pääosin samaa lähdekoodia pitäen sisällään lisäksi toteutuksen kontaktitietojen lisäykselle, mikä selittää osittain näiden kahden projektin välisen eron koodirivien määrässä. Jos Sencha Touch -projekti olisi sisältänyt kaiken vaadittavan toiminnallisuuden, sen mittaustulos olisi hyvin todennäköisesti ollut lähes tasoissa PhoneGap-projektin kanssa. Tämän vuoksi kärkisija päätettiin jakaa PhoneGap- ja Sencha Touch -projektien kesken.



Kuva 6.3: *Koodirivien lukumäärä eri kehitystyökaluilla toteutetussa esimerkkisovelluksessa*

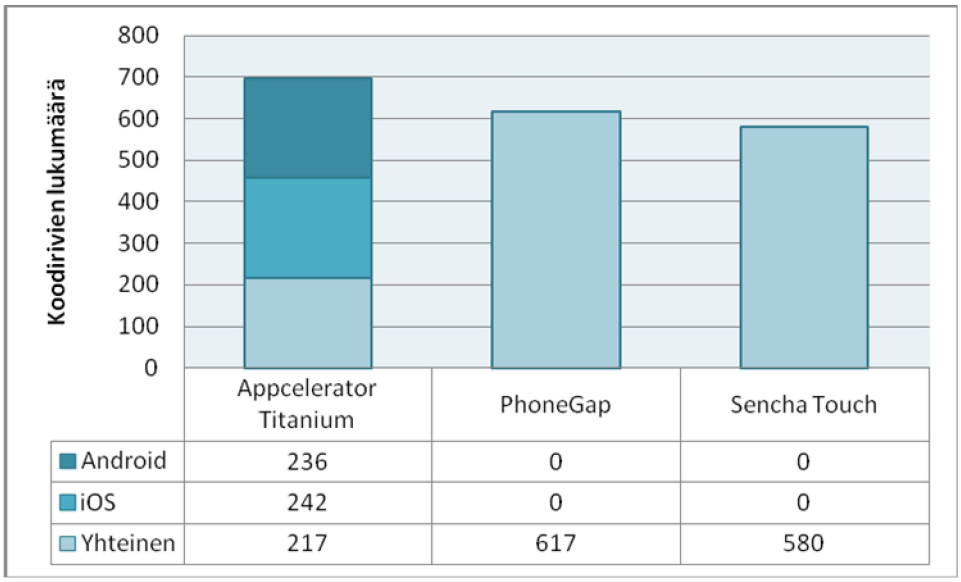
Tulos: PhoneGap 2,5 pistettä, Sencha Touch 2,5 pistettä, Appcelerator Titanium 1 piste.

6.1.4. Koodin uudelleenkäytettävyys

Ohjelmistokehysten ilmaisuvoimaisuuden lisäksi haluttiin verrata kuinka hyvin eri kehitystyökaluilla tuotettua lähdekoodia saadaan hyödynnettyä eri kohdealustojen välillä. Erityisesti kohdealustojen määrän noustessa koodin uudelleenkäytettävyyden merkitys kasvaa.

Alustakohtaisen koodin käyttö pyrittiin minimoimaan kaikissa projekteissa, jos se ei ollut esimerkkisovelluksen toteutuksen kannalta välttämätöntä. Web-käyttöliittymäelementtejä hyödyntävien PhoneGap- ja Sencha Touch -projektien kohdalla tämä oli mahdollista, mutta Appcelerator Titaniumin käyttämät natiivit käyttöliittymäelementit vaativat alustakohtaisia muokkauksia. Osittain CLOC-työkalun avulla kerätyt ja osittain käsin lasketut mittaustulokset on esitetty kuvassa 6.4.

Appcelerator Titanium -projektissa alustakohtaisen koodin määrä oli hyvin korkea, yli kaksi kolmasosaa kokonaiskoodimäärästä. Tätä selittää esimerkkisovelluksen käyttöliittymäkeskeisyys sekä valittu lähestymistapa alustakohtaisen koodin jäsentelyyn. Titanium -projektissa suurin osa Android- ja iOS-alustakohtaisesta koodista päätettiin jakaa omiin tiedostoihinsa, vaikka korkeamman koodin uudelleenkäytön olisi voinut saavuttaa käyttäen if-else-ehtolauseita. Yletön ehtolauseiden käyttö olisi kuitenkin saattanut heikentää koodin luettavuutta merkittävästi.



Kuva 6.4: *Alustakohtaisten (Android, iOS) ja alustariippumattomien (yhteinen) koodirivien lukumäärä eri kehitystyökaluilla toteutetussa esimerkkisovelluksessa*

Tulos: PhoneGap 2,5 pistettä, Sencha Touch 2,5 pistettä, Appcelerator Titanium 1 piste.

6.1.5. Lisenssi

Kehitystyökalun lisenssimalli määrittää sen käytöstä aiheutuvat kustannukset. Lisenssimaksuista aiheutuvia kustannuksia ei pidetty olennaisina vielä siinä vaiheessa, kun työkalut valittiin testattavaksi. Tällöin vaatimuksena oli ainoastaan, että työkalua pääsisi testaamaan ilmaiseksi vähintään 30 päivän kokeilujakson verran. Harkittaessa työkalun ottamista oikeasti käyttöön siitä aiheutuviin kustannuksiin on syytä tutustua. Taulukossa 6.1 on esitetty testattujen työkalujen lisenssimallit.

Taulukko 6.1: *Testattujen kehitystyökalujen lisenssimallit (Appcelerator Pricing, PhoneGap License, Sencha Touch Bundle Licensing)*

Kehitystyökalu	Lisenssimalli
Appcelerator Titanium	Avoin lähdekoodi (Apache 2.0)
PhoneGap	Avoin lähdekoodi (Apache 2.0)
Sencha Touch Bundle	Kaupallinen (Kehittäjäkohtainen maksullinen lisenssi)

Testatuista kehitystyökaluista Appcelerator Titaniumin ja PhoneGapin käyttö oli avoimen lähdekoodin myötä täysin ilmaista, kun taas Sencha Touch Bundle vaati 30 päivän kokeilujakson jälkeen kaupallisen lisenssin lunastamista. Lisenssimaksujen suuruusjärjestys oli työkalujen välillä siten täysin selvä, eikä hinnoittelua tarvinnut tutkia enää tarkemmin. Huomionarvoista oli kuitenkin, että Sencha Touch -JavaScript-kehys ja Sencha Cmd -web-natiivikäärijä olivat saatavilla Sencha Touch Bundlen ulkopuolelta erillisinä komponentteina myös avoimen lähdekoodin lisensseillä.

Tulos: Appcelerator Titanium 2,5 pistettä, PhoneGap 2,5 pistettä, Sencha Touch 1 piste.

6.2. Laadulliset arviointikriteerit

6.2.1. Työkalun opittavuus

Opittavuudella tarkoitetaan tässä sitä työmäärää, joka vaaditaan, että kehittäjä kykenee käyttämään itselleen uutta kehitystyökalua vähintään ennalta määritetyllä vähimmäistasolla. Täksi vähimmäistasoksi määritettiin esimerkkisovelluksen toteuttaminen kunkin työkalun avulla. Vähimmäistason saavuttamiseen vaadittua työmäärää arvioitiin kehittäjän subjektiivisten kokemusten ja esimerkkisovelluksen toteuttamiseen kuluneiden työpäivien määrän avulla.

Testatuista työkaluista Appcelerator Titanium ja Sencha Touch olivat PhoneGapista poiketen kokonaisvaltaisia mobiilisovellusten kehitysympäristöjä, minkä vuoksi vähimmäistason saavuttaminen vaati niillä enemmän ponnisteluja. Aikaa työkalun käyttöönotosta valmiiseen, molemmilla kohdealustoilla toimivaan esimerkkisovellukseen kului Appcelerator Titaniumilla arviolta noin seitsemän työpäivän ja Sencha Touchilla noin yhdeksän työpäivän verran. PhoneGapin avulla käärityn esimerkkisovelluksen valmistumiseen kului aikaa ainoastaan yhden työpäivän verran.

On kuitenkin otettava huomioon, että merkittävin osa ajasta Appcelerator Titanium ja Sencha Touch -projekteissa kului uusien ohjelmointirajapintojen opetteluun ja itse ohjelmointiin, kun taas PhoneGap-projektissa suurin osa ohjelmakoodista tuotiin Sencha Touch -projektista. PhoneGap ei täten ollut vertailukelpoinen kahden muun testatun kehitystyökalun kanssa, jos opittavuuden mittana käytettiin pelkästään esimerkkisovelluksen toteuttamiseen kulunutta aikaa kyseisellä kehitystyökalulla.

Arvioitaessa työkalujen opittavuutta subjektiivisesti kehittäjän aiemmilla kokemuksilla vastaavanlaisista toteutustekniikoista huomattiin olevan merkittävä vaikutus. Arvioinnit tehneellä kehittäjällä oli enemmän aiempaa kokemusta natiivisovellusten kuin web-sovellusten kehittämisestä mobiililaitteille, minkä vuoksi hän koki natiiveja käyttöliittymäelementtejä hyödyntävän Appcelerator Titanium -työkalun olleen helpommin lähestyttävä kuin web-elementtejä hyödyntävän Sencha Touch -työkalun. Lisäksi esimerkkisovelluksen virheiden jäljitys tuntui luontevammalta Titanium Studio -kehitysympäristössä kuin selainten kehittäjätyökalujen avulla.

Sencha Touch -työkalu ei ollut sekään hankala omaksua, mutta sekä Architect-kehitysympäristön käyttö että HTML5-käyttöliittymän ohjelmointi pelkän JavaScriptin avulla tuntui hyvin erilaiselta verrattuna aikaisempiin kokemuksiin vastaavista kehitysympäristöistä ja web-ohjelmoinnista. PhoneGap-projektissa käytettiin samoja kehitys-

ympäristöjä kuin natiivisovelluskehityksessä, mikä helpotti työkalun omaksumista kyseiset ympäristöt tuntevan kehittäjän näkökulmasta.

PhoneGap-projektin osalta arvioitiin tässä sitä työmäärää, joka vaadittiin, että Sencha Touch -kehyksellä tehty web-sovellus saatiin käärittyä PhoneGapin avulla hybridisovellukseksi verrattuna siihen, että vastaava tehtiin Senchan oman työkalun avulla. PhoneGap koettiin tässä tapauksessa lisäaskeleeksi. Sen avulla mobiilisovelluksia rakentavan kehittäjän onkin opeteltava PhoneGapin lisäksi jonkin käyttöliittymä- ja sovelluslogiikkakirjaston käyttö, mikä huomioitiin tämän vertailun tuloksissa.

Tulos: Appcelerator Titanium 3 pistettä, Sencha Touch 2 pistettä, PhoneGap 1 piste

6.2.2. Työkalun dokumentaation kattavuus

Työkalujen dokumentaation kattavuutta arvioitiin tutustumalla niiden kotisivuilla tarjottuihin ohjelmointirajapintadokumentteihin ja oppaisiin. Kattavuudella tarkoitetaan tässä sitä, kuinka laajamittaisesti työkalun dokumentaatio esittelee työkalun tarjoamat toiminnallisuudet. Hyvän kattavuuden lisäksi dokumentaation on oltava myös ajan tasalla, jotta siitä olisi hyötyä kehittäjälle. Puutteellinen tai vanhentunut dokumentaatio haittaa uuden työkalun käyttöönottoa ja työkalun avulla tehtävää kehitystyötä.

PhoneGap tarjosi kotisivuillaan ohjelmointirajapintadokumentaation lisäksi yleisoppaita muun muassa sen käyttöönottoa ja liitännäisten luomista varten. Ohjelmointirajapintadokumentit oli jaettu selkeästi PhoneGapin tukemien laiteominaisuuksien mukaan. Dokumenttisivut sisälsivät useimmiten lyhyen ja pitkän esimerkin kunkin esitetyn metodin käytöstä. Ne esittelivät usein myös katsauksen tunnettuihin alustakohtaisiin oikkuihin (*engl. quirk*). PhoneGapin ohjelmointirajapintoja käytettiin hyvin rajatusti projektin aikana, mutta yleiskuva sen dokumentaation kattavuudesta ja ajantasaisuudesta oli hyvin positiivisen.

Appceleratorin tarjoama dokumentaatio oli jaettu ohjelmointirajapintadokumentteihin, oppaisiin ja videoihin (Appcelerator Titanium Docs). Luettavia oppaita oli muihin työkaluihin nähden runsaasti tarjolla. Niiden käsittelemät aiheet vaihtelivat Titaniumin käyttöönotosta valmiiden sovellusten jakeluun eri kohdealustoille, joten niistä oli apua koko toteutusprosessin ajan.

Titaniumin ohjelmointirajapintoja esittelevä dokumentaatio oli jäsennetty puurakenteeseen, jonka sai järjestettyä joko luokkien nimiavaruuden tai periytymissuhteen mukaan. Jäsentely osoittautui toimivaksi, koska rajapintoja oli tarjolla huomattavasti enemmän kuin PhoneGapissa. Dokumenttisivut sisälsivät useimmiten vain lyhyen esimerkin esitetyn metodin käytöstä, minkä vuoksi ohjelmointirajapintadokumentaation voitiin katsoa olleen ainakin siltä osin PhoneGapia suppeampi. Video-oppaisiin ei juurikaan tutustuttu, koska suurin osa niistä ei ollut muun dokumentaation tavoin ajan tasalla.

Sencha käytti samaa dokumenttien julkaisujärjestelmää kuin Appcelerator. Tämä julkaisujärjestelmä osoittautui käteväksi, koska se osasi tallentaa sen sisällä avatut välilehdet selaimen muistiin seuraavaa vierailukertaa varten. Appceleratorin tavoin Sencha tarjosi Touch-kehykselleen ohjelmointirajapintadokumentteja, oppaita ja videoita. Niiden lisäksi tarjolla oli myös selaimella ajettavia esimerkkisovelluksia, jotka olivat hyödyllisiä etenkin, jos halusi nähdä miten erilaiset käyttöliittymäelementit toimivat.

Appcelerator Titaniumin dokumentaatioon verrattuna Sencha Touch -kehyksen dokumentaatio oli suppeaa. Oppaita oli Titanium-kehystä selvästi vähemmän ja ne käsittelivät lähinnä perustoimintoja. Osa ohjelmointirajapintadokumenteista vaikutti lisäksi keskeneräisiltä, koska niistä puuttui kuvaustekstit ja esimerkit kokonaan. Sencha tarjosi Architect-kehitysympäristöä varten omat oppaansa, mutta niistä vain pieni osa käsitteli Architectin käyttöä Sencha Touch -kehyksen kanssa.

PhoneGapin dokumentaation katsottiin olleen ainakin sen avulla toteutettujen toiminnallisuuksien osalta muita työkaluja kattavampi ja paremmin ajan tasalla. On kuitenkin huomioitava, että PhoneGapin dokumentaation ylläpitäminen ei ole suurempiin Appcelerator Titanium- ja Sencha Touch -kehyksiin verrattuna yhtä työlästä, koska se on keskittynyt ainoastaan laiteominaisuuksia hyödyntäviin toiminnallisuuksiin.

Tulos: PhoneGap 3 pistettä, Appcelerator Titanium 2 pistettä, Sencha Touch 1 piste

6.2.3. Esimerkkisovelluksen onnistuminen

Esimerkkisovelluksen onnistumista arvioitiin sen mukaan, kuinka hyvin sille määritetyt toiminnallisuudet saatiin toteutettua Android- ja iOS-kohdealustoille kullakin kehitystyökalulla. Arvioinnit suoritettiin Samsung Galaxy S3 - ja iPad 3 -testilaitteilla, jotka yhdistettiin WiFi-yhteydellä Metson sisäverkkoon ennen esimerkkisovellusten käyttöä. Testilaitteissa kytkettiin lisäksi GPS-paikannus päälle.

Jokaisella esimerkkisovelluksella tehtiin viisi asiakashakua etukäteen määritetyillä hakusanoilla. Näiden hakujen hakutuloksista valittiin jokaisesta yksi asiakas, jonka avulla testattiin esimerkkisovelluksen muita toiminnallisuuksia: asiakkaan tallennusta suosikiksi laitteen lokaaliin tietokantaan, asiakkaan tietojen lisäystä osoitekirjaan sekä asiakkaan ja käyttäjän oman sijainnin esittämistä kartalla.

Sencha Touch ei tukenut lainkaan kontaktitietojen lisäystä osoitekirjaan, mikä oli tiedossa jo ennen varsinaista esimerkkisovelluksen toteutusprosessia. Kyseistä toiminnallisuutta ei voitu siten toteuttaa Sencha Touchin avulla rakennettuun sovellukseen, minkä katsottiin olleen merkittävin yksittäinen puute toiminnallisuuksissa eri kehitystyökaluilla rakennettujen esimerkkisovellusten välillä. Toinen Sencha Touch -esimerkkiso-

velluksessa havaittu puute ilmeni vain Android-testilaitteella, jolla ei saatu paikannettua käyttäjän omaa sijaintia.

Appcelerator Titaniumin avulla luodussa esimerkkisovelluksessa ainoa ongelma havaittiin geokoodauksessa, eli katuosoitteen muuntamisessa kartalla esitettäviksi koordinaateiksi. Palvelu näytti palauttavan koordinaatteja onnistuneesti ainoastaan Suomessa oleville Metson asiakkaille. Tämä johtui ainakin osittain siitä, että Titanium ei Sencha Touchista poiketen käyttänyt tarjoamissaan geolokaatio-funktioissa palveluntarjoajana Googlea, vaan MapQuest Nominatim Search -nimistä joukkoistamisen avulla toimivaa palvelua (Appcelerator Titanium Docs).

PhoneGapin avulla rakennettu esimerkkisovellus oli ainoa, jossa ei havaittu yhtään puutetta testatuissa toiminnallisuuksissa. Tämän vuoksi se todettiin toiminnallisuuksien kannalta parhaiten onnistuneeksi sovellukseksi.

Tulos: PhoneGap 3 pistettä, Appcelerator Titanium 2 pistettä, Sencha Touch 1 piste

6.2.4. Esimerkkisovelluksen käyttöliittymän sulavuus

Käyttäjän antamien syötteiden vasteajat ja animaatioiden sulavuus vaikuttavat siihen miten mielekkääksi käyttäjä kokee sovelluksen käytön. Pahimmassa tapauksessa hidasteleva käyttöliittymä voi vaikuttaa käyttäjän mielipiteeseen sovelluksesta niin paljon, että hän päättää lopettaa sen käytön kokonaan. Esimerkkisovelluksen käyttöliittymän sulavuutta arvioitiin silmämääräisesti molemmilla kohdealustoilla. Vertailukohtana käytettiin natiivisovelluksia, joiden etu tulkattaviin sovelluksiin ja hybridisovelluksiin nähden on niitä vahvempi suorituskyky.

PhoneGap- ja Sencha Touch -projekteissa käytettiin molemmissa Sencha Touch -kehyksen avulla rakennettua web-käyttöliittymää. Hybridisovellusten suorituskyky on riippuvainen toteutustekniikoiden ja laitteiston suorituskyvyn lisäksi kohdealustan selainmoottorin suorituskyvystä ja sen tukemista HTML5-ominaisuuksista. PhoneGap- ja Sencha Touch -työkaluilla käärittyjen esimerkkisovelluksen käyttö sujui testilaitteilla pääosin ilman merkittäviä viiveitä käyttöliittymässä. Natiivisovelluksiin verrattaessa pieni viive oli kuitenkin kokoajan havaittavissa erityisesti painikkeissa, jotka reagoivat painalluksiin joitain kymmenesosasekunteja hitaammin. Myös kartan vierittäminen ja suurentaminen tuntui hitaammalta ja vähemmän sulavalta kuin natiivisovelluksissa.

Appcelerator Titanium -projektissa esimerkkisovelluksen käyttöliittymä rakennettiin käyttäen kohdealustan omia natiivielementtejä Titaniumin tarjoamien JavaScript-ohjelmointirajapintojen kautta. Tällöin Titanium-sovelluksen käyttöliittymä vastasi ulkonäöltään natiivisovelluksen käyttöliittymää. Natiivisovelluksista poiketen kutsuja oli kuitenkin hidastamassa Titaniumin JavaScript-ajoympäristö, joka muodosti ylimääräisen kerroksen kutsuvan koodin ja kohdealustan käyttöliittymäkirjastojen välille. Tästä

hidasteesta huolimatta Titaniumin avulla luodun sovelluksen käyttöliittymässä ei havaittu lainkaan silmiinpistäviä viiveitä tai tökkimistä kummallakaan testilaitteella, minkä vuoksi Titanium valittiin tämän vertailun voittajaksi. Kuvakaappaukset esimerkkisovellusten käyttöliittymänäkymistä ovat liitteessä A.

Tulos: Appcelerator Titanium 3 pistettä, PhoneGap 1,5 pistettä, Sencha Touch 1,5 pistettä

6.3. Yhteenveto

Yleisesti ottaen kaikki testatut kehitystyökalut suoriutuivat testijaksosta varsin hyvin. Missään testatussa kehitystyökalussa ei havaittu toteutusprosessin aikana yhtään sellaista puutetta, joka olisi estänyt esimerkkisovelluksen kehittämisen loppuun saakka. Sencha Touchin laiteominaisuuksille tarjoamat ohjelmointirajapinnat eivät tosin mahdollistaneet kontaktitietojen lisäystä osoitekirjaan, mikä oli tiedossa jo valittaessa testattavia kehitystyökaluja. Kaikki testatut kehitystyökalut tuntuivat testin perusteella valmiilta täysimittaisen mobiilisovelluksen kehitystä varten.

Esimerkkisovelluksen kehitysvauhdin arvioitiin olleen kehittäjän aiempiin kokemuksiin vedoten selvästi nopeampi alustariippumattomilla kehitystyökaluilla kuin, jos sama sovellus olisi toteutettu käyttäen molempien kohdealustojen virallisia kehitystyökaluja. Testattujen alustariippumattomien kehitystyökalujen oppimiskynnystä pidettiin myös huomattavasti kohdealustojen virallisia työkaluja matalampana. Vastaavan osaamistason saavuttamiseen, mihin kohdealustojen virallisilla kehitystyökaluilla kului joitakin viikkoja, meni testatuilla alustariippumattomilla työkaluilla joitakin päiviä.

Appcelerator Titanium saavutti vertailussa muita testattuja kehitystyökaluja suuremman yhteispistemäärän (taulukko 6.2). Titaniumin avulla rakennettujen tulkattavien sovellusten vahvuuksina PhoneGapin ja Sencha Touchin avulla käärittyihin hybridisovelluksiin nähden pidettiin vähäisempää keskusmuistin käyttöä ja sulavampaa käyttöliittymää. Titaniumin oppimiskynnyksen koettiin lisäksi olleen natiivisovelluskehitystä entuudestaan tuntevan ohjelmistokehittäjän näkökulmasta web-tekniikoihin keskittyneitä PhoneGap- ja Sencha Touch -kehitystyökaluja matalampi.

Hybridisovellusten merkittävin etu Titaniumin avulla rakennettuihin tulkattaviin sovelluksiin nähden oli koodin täydellinen uudelleenkäytettävyys kohdealustojen välillä. Toisin kuin Titanium-sovellusten, joiden käyttöliittymä rakennettiin käyttäen alustakohtaisia natiivikäyttöliittymäelementtejä, hybridisovellusten web-käyttöliittymä voitiin rakentaa yhtenäiseksi eri kohdealustoille käyttämättä lainkaan alustakohtaista koodia. Tämä etu kääntyi kuitenkin haitaksi arvioitaessa esimerkkisovelluksen käyttöliittymän sulavuutta, koska hybridisovellusten web-käyttöliittymän koettiin reagoineen käyttäjän syötteisiin ajoittain Titanium-sovelluksen natiivikäyttöliittymää hitaammin.

Taulukko 6.2: *Testattujen kehitystyökalujen arviointikriteereistä saavuttamat pistemäärät*

	Appcelerator Titanium	PhoneGap (+ Sencha Touch)	Sencha Touch
Sovelluksen asennuspaketin koko	2	1	3
Sovelluksen keskusmuistin käyttö	3	1	2
Koodirivien yhteismäärä	1	2,5	2,5
Koodin uudelleenkäytettävyys	1	2,5	2,5
Lisenssi	2,5	2,5	1
Työkalun opittavuus	3	1	2
Työkalun dokumentaation kattavuus	2	3	1
Esimerkkisovelluksen onnistuminen	2	3	1
Esimerkkisovelluksen käyttöliittymän sulavuus	3	1,5	1,5
Yhteensä	**19,5**	**18**	**16,5**

Sencha Touch suoriutui vertailussa heikoiten, koska se ei tukenut laiteominaisuuksia yhtä kattavasti kuin Appcelerator Titanium ja PhoneGap. Tämän lisäksi sen dokumentaatiossa havaittiin eniten puutteita, minkä vuoksi se myös tuntui muita vertailun kehitystyökaluja keskeneräisemmältä. Web-käyttöliittymä- ja sovelluslogiikkakirjastona Sencha Touch muodosti kuitenkin yhdessä PhoneGapin kanssa hyvin toimivan kokonaisuuden. PhoneGapin avulla kääritty esimerkkisovellus arvioitiin toiminnallisuuksien kannalta vertailun onnistuneimmaksi. Piste-ero vertailussa suurimman ja toiseksi suurimman yhteispistemäärän saaneiden Appcelerator Titaniumin ja PhoneGapin välillä oli pieni – 1,5 pistettä.

6.4. Jatkotutkimuksen kohteet

Tähän on koottu tutkimuksen aikana ja sen jälkeen mieleen tulleita tutkimusaiheita, joita ei kuitenkaan haluttu tai ehditty sisällyttää tämän tutkimuksen laajuuteen. Nämä esille tulleet tutkimusaiheet ovat siten potentiaalisia kohteita jatkotutkimukselle:

- **Tietoturvallisuus:** Vaikka esimerkkisovelluksessa esitetty asiakastieto ei ollutkaan turvaluokitukseltaan kovin kriittistä, tietoturvallisuuden arviointi on yksi tärkeimmistä jatkotutkimuskohteista, jos jokin testatuista kehitystyökaluista halutaan ottaa oikeaan käyttöön. Tällöin on tutkittava muun muassa kuinka laitteen muistiin tallennettu tieto salataan, jotta asiattomat eivät pääse siihen käsiksi.

- **Testaus useammalla testilaitteella:** Tutkimuksen aikana käytössä oli ainoastaan kaksi testilaitetta: yksi Android- ja yksi iOS-laite. Erityisesti esimerkkisovelluksen toimivuutta ja käyttöliittymän sulavuutta olisi ollut järkevä testata useammalla kuin yhdellä testilaitteella ja käyttöjärjestelmäversiolla per kohdealusta. Testilaitteita ei ollut kuitenkaan tällöin enempää saatavilla, joten testaus suorituskyvyltään eritasoisilla ja eri käyttöjärjestelmäversiolla varustetuilla testilaitteilla päätettiin siirtää myöhemmäksi.
- **Suorituskykymittaukset:** Esimerkkisovelluksen suorituskykyä olisi voinut vertailla nykyistä laajemmin esimerkiksi mittaamalla sovelluksen käynnistymisnopeutta ja suorittamalla rasitustestejä suurilla datamäärillä.
- **Käytettävyys- tai käyttäjäkokemusarviot:** Asiantuntijan tekemät käytettävyys- tai käyttäjäkokemusarviot eri kehitystyökalujen avulla rakennetuista sovelluksista olisivat olleet mielenkiintoinen lisä kehittäjän itse tekemille subjektiivisille arvioille.

7. JOHTOPÄÄTÖKSET

Tässä diplomityössä tutustuttiin alustariippumattoman mobiilisovelluskehityksen tekniikoihin sekä pyrittiin löytämään Metson MAC-segmentin mobiilisovelluskehityksestä vastaavan GAD-tiimin käyttötarkoituksiin parhaiten soveltuva alustariippumaton kehitystyökalu. Tarkempaan testiin valittujen Appcelerator Titanium-, PhoneGap- ja Sencha Touch -kehitystyökalujen avulla rakennettiin tiimin käyttötarpeita vastaava esimerkkisovellus, jonka toteutusprosessia ja lopputulosta vertailtiin yhdeksän arviointikriteerin perusteella. Esimerkkisovelluksen kohdealustoiksi valittiin Android- ja iOS-mobiilikäyttöjärjestelmät.

Tutkimustulosten perusteella voidaan todeta, että alustariippumattomat mobiilisovelluskehitystyökalut ovat varteen otettava vaihtoehto virallisille alustakohtaisille kehitystyökaluille. Niiden oppimiskynnys on alustakohtaisia kehitystyökaluja matalampi, koska niiden peruskäytön opettelu vie aikaa useiden viikkojen sijaan nopeimmillaan päiviä. Kehitysvauhti erityisesti tuettaessa useaa mobiilialustaa on huomattavasti nopeampi, koska samaa lähdekoodia voidaan hyödyntää parhaassa tapauksessa täydellisesti kohdealustojen välillä. Tämän lisäksi sovellukset voidaan useimmiten testata ja pakata saman kehitysympäristön avulla.

Alustariippumattomien kehitystyökalujen suurimmat heikkoudet verrattuna alustojen virallisiin kehitystyökaluihin ovat niitä rajatummat laiteominaisuuksille tarjotut ohjelmointirajapinnat sekä myöhäisempi pääsy alustojen uusiin toiminnallisuuksiin. Kaikki alustariippumattomat kehitystyökalut eivät edes pyri tukemaan kaikkia alustojen toiminnallisuuksia, jos niitä ei ole saatavilla kaikilla sen tukemilla kohdealustoilla, parantaakseen koodin uudelleenkäyttöä alustojen välillä.

Hybridisovellukset ja tulkattavat sovellukset ovat suorituskyvyltään jäljessä natiivisovelluksia, mikä johtuu niiden käyttämän ajoympäristön lisäämästä ylimääräisestä kerroksesta ohjelmakoodin ja sitä suorittavan laitteiston välissä. Testijakson aikana vaihtelevaa suorituskykyä havaittiin erityisesti hybridisovelluksissa, joiden käyttöliittymä reagoi välillä hitaasti käyttäjän syötteisiin.

Tässä tutkimuksessa tehdyssä kolmen alustariippumattoman kehitystyökalun vertailussa suurimman yhteispistemäärän sai Appcelerator Titanium niukasti ennen toiseksi tullutta PhoneGapia. Appcelerator Titaniumin avulla rakennettujen tulkattavien sovellusten etu-

na PhoneGapin ja Sencha Touchin avulla käärittyihin hybridisovelluksiin nähden pidettiin erityisesti niitä alhaisempaa keskusmuistin käyttöä ja sulavampaa käyttöliittymää.

Huomattavin ero kehitystyökaluilla rakennettujen esimerkkisovellusten välillä oli niiden käyttöliittymien ulkonäössä ja toteutustekniikoissa. Titanium-sovellusten käyttöliittymä oli rakennettu käyttäen alustakohtaisia natiivikäyttöliittymäelementtejä, kun taas hybridisovellusten web-käyttöliittymä oli toteutettu HTML5-tekniikoilla Sencha Touch -JavaScript-kehystä hyödyntäen. Vaikka hybridisovellusten web-käyttöliittymä osoittautui Titanium-sovelluksen natiivikäyttöliittymää hitaammaksi, voitiin se rakentaa yhtenäiseksi molemmille tuetuille kohdealustoille, jolloin alustakohtaista koodia ei tarvinnut käyttää lainkaan. Täydellinen koodin uudelleenkäytettävyys alustojen välillä olikin hybridisovellusten suurin vahvuus verrattuna Titanium-sovelluksiin, joiden käyttöliittymää jouduttiin mukauttamaan alustakohtaisesti.

Mitä mobiilisovelluskehitystyökaluja Metson GAD-tiimin tulisi tulevaisuudessa käyttää, riippuu paljon siitä, minkälaisia sovelluksia niiden avulla halutaan rakentaa ja mitä kohdealustoja halutaan tukea. Monimutkaiset, suorituskyvyn ja laiteominaisuuksien kannalta kriittiset sovellukset on edelleen järkevintä toteuttaa käyttäen alustakohtaisia kehitystyökaluja edellä esitettyjen alustariippumattomien kehitystyökalujen heikkouksien vuoksi. Tuettavien kohdealustojen määrän kasvaessa sekä suorituskyky- ja laiteominaisuusvaatimusten madaltuessa on kuitenkin syytä harkita alustariippumattoman kehitystyökalun käyttöönottoa.

Vertailun suurimman pistemäärän saanut Appcelerator Titanium olisi varteenotettavin valinta kehitystyökaluksi, jos tuettavat alustat olisivat ainoastaan Android- ja iOS. Jos alustatuen tulisi olla tätä laajempi, järkevämpi valinta olisi vertailussa toiseksi tullut PhoneGap-kehitystyökalun ja Sencha Touch -JavaScript-kehyksen yhdistelmä Titaniumia laajemman kohdealustatuen ja paremman koodin uudelleenkäytettävyyden vuoksi. Sencha Touch -JavaScript-kehyksen avulla rakennettua sovellusta voidaan ajaa palvelimella myös web-sovelluksena, jos pääsy PhoneGapin tarjoamiin laiterajapintoihin ei ole sovelluksen toiminnan kannalta välttämätöntä.

Alustariippumattomien kehitystyökalujen suosion voidaan odottaa kasvavan tulevaisuudessa mobiililaitteiden määrän kasvaessa ja mobiilikäyttöjärjestelmien markkinaosuuksien tasoittuessa. Androidin ja iOS:n saadessa enemmän varteenotettavia haastajia myös aiemmin alustakohtaisissa kehitystyökaluissa pitäytyneiden kehittäjien on mietittävä uudelleen, miten he aikovat tukea yhä useampia alustoja.

Gartnerin ennusteiden (Rivera & Van der Meulen 2013) mukaan vuoteen 2016 mennessä yli puolet yrityskäyttöön suunnatuista mobiilisovelluksista on hybridisovelluksia. Hybridisovelluksille povataan kasvua jo olemassa olevan vankan web-sovelluskehittäjäkunnan keskittyessä entistä enemmän mobiilisovellusten kehittämiseen. Tämän

tutkimuksen aikana havaitut hybridisovelluksia haitanneet suorituskykyongelmat saattavatkin olla jo lähitulevaisuudessa historiaa mobiililaitteiden laitteiston suorituskyvyn kasvaessa jatkuvasti.

LÄHTEET

Adobe AIR Dev Center [WWW]. [viitattu 6.6.2013]. Saatavissa: http://www.adobe.com/devnet/air.html

Anderson, J. 2013. Appcelerator Titanium: Up and Running. O'Reilly. 140 p.

Android Developer [WWW]. [viitattu 24.5.2013]. Saatavissa: http://developer.android.com/index.html

Appcelerator Pricing [WWW]. [viitattu 10.7.2013]. Saatavissa: http://www.appcelerator.com/plans-pricing/

Appcelerator Titanium Docs [WWW]. [viitattu 22.10.2013]. Saatavissa: http://docs.appcelerator.com/titanium/latest/

Apple iOS [WWW]. [viitattu 22.2.2013]. Saatavissa: http://www.apple.com/fi/ios/

Apple Press Releases [WWW]. [viitattu 1.3.2013]. Saatavissa: http://www.apple.com/pr/library/

AppMakr [WWW]. [viitattu 5.5.2013]. Saatavissa: http://www.appmakr.com/

Brown, E. 2012. 5 Mobile Linux OSes that Dare to Compete with Android [WWW]. Linux.com. 10.10.2012. [viitattu 14.2.2013]. Saatavissa: https://www.linux.com/news/embedded-mobile/mobile-linux/648324-5-mobile-linux-oses-that-dare-to-compete-with-android

Charland, A., & Leroux, B. 2011. Mobile application development: web vs. native. Communications of the ACM, 54(5), pp. 49-53.

CLOC [WWW]. [viitattu 24.9.2013]. Saatavissa: http://cloc.sourceforge.net/

Corona [WWW]. [viitattu 9.11.2013]. Saatavissa: http://www.coronalabs.com/products/corona-sdk/

Feloney, S. 2013. Titanium Support Plans for Windows 8. Appcelerator Developer Blog. 17.1.2013. [viitattu 15.7.2013]. Saatavissa: http://developer.appcelerator.com/blog/2013/01/titanium-support-plans-for-windows-8.html

Friedman, L. 2011. Hands on with iOS over-the-air updates [WWW]. Macworld.com. 10.11.2011. [viitattu 6.4.2013]. Saatavissa: http://www.macworld.com/article/1163526/hands_on_with_ios_over_the_air_updates.html

Heitkötter, H., Hanschke, S. & Majchrzak, T. A. 2013. Evaluating cross-platform development approaches for mobile applications. In Proceedings of the 8th WEBIST. pp. 299-311.

HTML5test.com [WWW]. [viitattu 19.4.2013]. Saatavissa: http://html5test.com/

iOS Developer Program [WWW]. [viitattu 22.2.2013]. Saatavissa: https://developer.apple.com/programs/ios/

iOS Technology Overview [WWW]. [viitattu 23.10.2013]. Saatavissa: http://developer.apple.com/library/ios/documentation/miscellaneous/conceptual/iphoneostechoverview/Introduction/Introduction.html

iOS Version Stats [WWW]. [viitattu 24.5.2013]. Saatavissa: http://david-smith.org/iosversionstats/

Jones, S., Voskolou, Voskoglou, C., Vakulenko, M., Measom, V., Constantinou A. & Kapetanakis, M. 2012. Cross-Platform Developer Tools 2012. VisionMobile, London. 97 p.

jQueryMobile [WWW]. [viitattu 10.5.2013]. Saatavissa: http://jquerymobile.com/

Juntunen, A., Jalonen, E. & Luukkainen, S. 2013. HTML 5 in Mobile Devices–Drivers and Restraints. 2013 46th Hawaii International Conference on System Sciences (HICSS). pp. 1053-1062.

Kokkonen, J. 2012. Windows Phone 8 [WWW]. TaskuMuro. 18.12.2012. [viitattu 24.7.2013]. Saatavissa: http://taskumuro.com/artikkelit/windows-phone-8

Kosmaczewski, A. 2013. Sencha Touch 2 Up and Running. O'Reilly. 259 p.

Krill, P. 2009. Appcelerator enables iPhone, Android app dev [WWW]. InfoWorld. 8.6.2009. [viitattu 8.10.2013]. Saatavissa: http://www.infoworld.com/d/developer-world/appcelerator-enables-iphone-android-app-dev-655

Llamas, R., Restivo, K. & Shirer, M. 2012. Worldwide Mobile Phone Growth Expected to Drop to 1.4% in 2012 Despite Continued Growth Of Smartphones [WWW]. International Data Corporation (IDC). 4.12.2012. [viitattu 14.2.2013]. Saatavissa: http://www.idc.com/getdoc.jsp?containerId=prUS23818212

Maia, C., Nogueira, L., & Pinho, L. M. 2010. Evaluating Android OS for Embedded Real-Time Systems. In Proceedings of the 6th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications, Brussels, Belgium. pp. 63-70.

Mainelli, T., Reith, R. & Shirer, M. 2013. Low Cost Products Drive Forecast Increases in the Tablet Market [WWW]. International Data Corporation (IDC). 12.3.2013. [viitattu 24.3.2013]. Saatavissa: http://www.idc.com/getdoc.jsp?containerId=prUS24002213

Marmalade [WWW]. [viitattu 9.11.2013]. Saatavissa: http://www.madewithmarmalade.com/

Mawston, N. 2013. Android and Apple iOS Capture a Record 92 Percent Share of Global Smartphone Shipments in Q4 2012 [WWW]. Strategy Analytics. 28.1.2013. [viitattu 8.2.2013]. Saatavissa: http://blogs.strategyanalytics.com/WSS/post/2013/01/28/Android-and-Apple-iOS-Capture-a-Record-92-Percent-Share-of-Global-Smartphone-Shipments-in-Q4-2012.aspx

McWherter, J. & Gowell, S. 2012. Professional Mobile Application Development. John Wiley & Sons. Indianapolis, Indiana, USA. 403 p.

Metso Avenue [Metso intranet]. [viitattu 29.5.2013].

Metso Notes Traveler DB [Excel-tiedosto]. [viitattu 29.5.2013]

Mikkonen, T., & Taivalsaari, A. 2011. Apps vs. Open Web: The Battle of the Decade. In Proceedings of the 2nd Workshop on Software Engineering for Mobile Application Development, pp. 22-26.

Mobile HTML5 [WWW]. [viitattu 19.4.2013]. Saatavissa: http://mobilehtml5.org/

Olson, P. 2013. 10 Predictions For The Mobile Industry In 2013 [WWW]. Forbes. 2.1.2013. [viitattu 14.2.2013]. Saatavissa: http://www.forbes.com/sites/parmyolson/2013/01/02/10-predictions-for-the-mobile-industry-in-2013/

Open Handset Alliance [WWW]. [viitattu 21.2.2013]. Saatavissa: http://www.openhandsetalliance.com

Paul, I. 2013. Google Chromium project leaves WebKit to work with Blink browser engine [WWW]. PCWorld. [viitattu 27.4.2013]. Saatavissa: http://www.pcworld.com/article/2033063/google-chromium-project-leaves-webkit-to-work-with-blink-browser-engine.html

PhoneGap [WWW]. [viitattu 17.5.2013]. Saatavissa: http://phonegap.com/

PhoneGap Build [WWW]. [viitattu 26.8.2013]. Saatavissa: http://build.phonegap.com/

PhoneGap Debug [WWW]. [viitattu 26.8.2013]. Saatavissa: http://debug.phonegap.com/

PhoneGap Features [WWW]. [viitattu 1.7.2013]. Saatavissa: http://phonegap.com/about/feature/

PhoneGap License [WWW]. [viitattu 7.10.2013]. Saatavissa: http://phonegap.com/about/license/

Qt [WWW]. [viitattu 9.11.2013]. Saatavissa: http://qt.digia.com/product/

Rivera, J. & Van der Meulen, R. 2013. Gartner Says by 2016, More Than 50 Percent of Mobile Apps Deployed Will be Hybrid [WWW]. Gartner. 4.2.2013. [viitattu 17.5.2013]. Saatavissa: http://www.gartner.com/newsroom/id/2324917

Sencha [WWW]. [viitattu 26.8.2013]. Saatavissa: http://www.sencha.com/

Sencha Touch Bundle Licensing [WWW]. [viitattu 26.8.2013]. Saatavissa: http://www.sencha.com/products/touch-bundle/license/

Sencha Touch Docs [WWW]. [viitattu 22.10.2013]. Saatavissa: http://docs.sencha.com/touch/2.2.1/

Singh, R., Bhargava, P., & Kain, S. 2008. Smart phones: A tutorial. Ubiquity, 2008, April, 4. Saatavissa: http://ubiquity.acm.org/article.cfm?id=1366322

Smith, J.E. & Nair, R. 2005. The Architecture of Virtual Machines. Computer, May 2005. pp. 32-38.

System Monitor [WWW]. Google Play -sovelluskauppa. [viitattu 24.9.2013]. Saatavissa: https://play.google.com/store/apps/details?id=com.p_soft.sysmon&hl=en

Thomas, D. 2013. Nokia to stop shipping Symbian smartphones [WWW]. The Financial Times. 11.6.2013. [viitattu 25.7.2013]. Saatavissa: http://www.ft.com/intl/cms/s/0/d614b7ba-cddc-11e2-a13e-00144feab7de.html

Trew, J. 2012. Microsoft confirms no upgrade path to Windows Phone 8, unveils 7.8 for legacy devices [WWW]. Engadget. 20.6.2012. [viitattu 26.7.2013]. Saatavissa: http://www.engadget.com/2012/06/20/microsoft-unveils-windows-phone-7-8-for-legacy-devices/

Unity [WWW]. [viitattu 9.11.2013]. Saatavissa: http://unity3d.com/unity

VisionMobile: Developer Economics 2013. VisionMobile, London. 60 p.

W3C: HTML5 differences from HTML4 [WWW]. [viitattu 14.4.2013]. Saatavissa: http://dev.w3.org/html5/html4-differences/

Wargo, J. M. 2012. PhoneGap Essentials: Building Cross-platform Mobile Apps. Addison-Wesley Professional. 359 p.

Wasserman, A. I. 2010. Software Engineering Issues for Mobile Application Development. Proc. FSE/SDP Workshop Future of Software Eng. Research (FoSER 10), ACM, pp. 397-400.

Weinre [WWW]. [viitattu 2.9.2013]. Saatavissa: http://people.apache.org/~pmuellr/weinre/docs/latest/

Winokur, D. 2011. Flash to Focus on PC Browsing and Mobile Apps; Adobe to More Aggressively Contribute to HTML5 [WWW]. Adobe. [viitattu 11.6.2013]. Saatavissa: http://blogs.adobe.com/conversations/2011/11/flash-focus.html

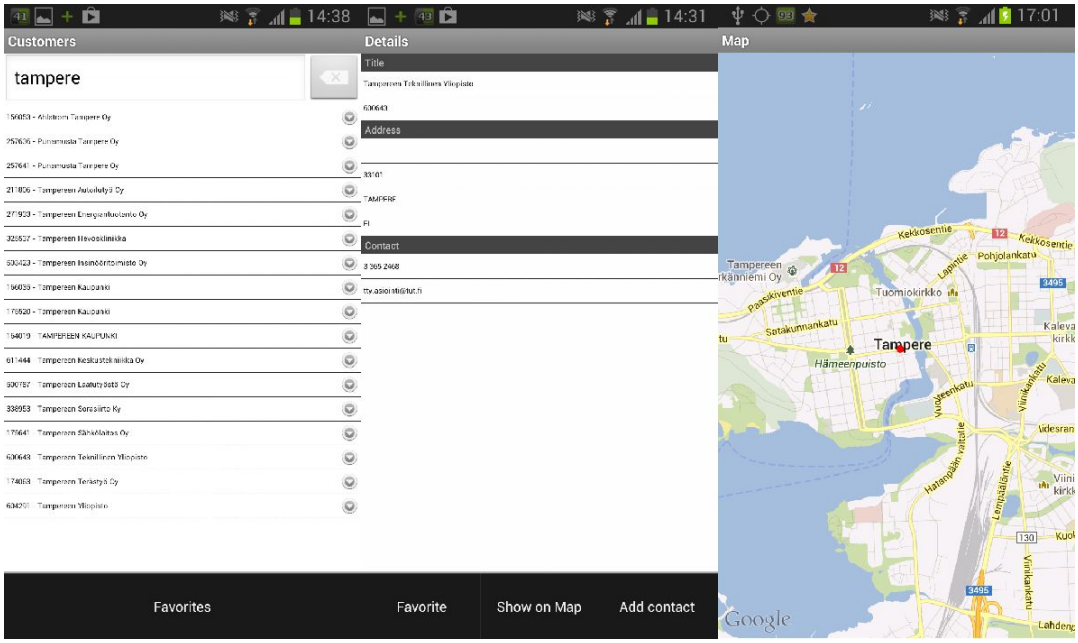
WP Developer [WWW]. [viitattu 24.7.2013]. Saatavissa: http://developer.windowsphone.com/

Xamarin Dev Center [WWW]. [viitattu 6.6.2013]. Saatavissa: http://docs.xamarin.com/

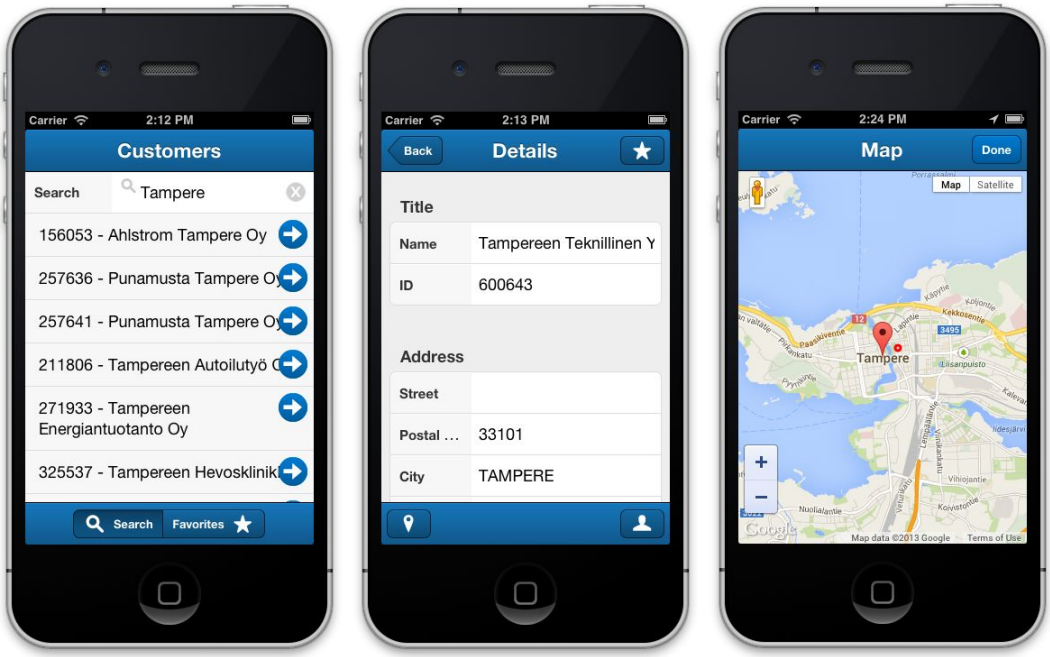
Xamarin How It Works [WWW]. [viitattu 5.7.2013]. Saatavissa: http://xamarin.com/how-it-works

LIITE A: KUVAKAAPPAUKSET

Appcelerator Titanium -kehitystyökalulla toteutettu esimerkkisovellus iOS-alustalla



Appcelerator Titanium -kehitystyökalulla toteutettu esimerkkisovellus Android-alustalla



Sencha Touch -kehitystyökalulla (+ PhoneGap-kehitystyökalulla) toteutettu esimerkki-sovellus iOS-alustalla (Android-alustalla käyttöliittymä oli identtinen)